DESARROLLO DE UN CONTROLADOR CANOPEN Y SU INTEGRACIÓN EN UN ROBOT MOVIL

UNIVERSIDAD CARLOS III DE MADRID
DEPARTAMENTO DE AUTOMÁTICA

Autor: CONRADO GRANDAL REVUELTA

Tutor: María Malfaz Vázquez
Director: Víctor González Pacheco

INGENIERIA INDUSTRIAL: ELECTRÓNICA Y AUTOMÁTICA
2012

Abstract

The Robotics Lab. Group's new mobile robot includes two motor motion controllers that command the wheels of that robot. These controllers are in charge of communicating the robot's motors with its main computer. Both computer and motion controllers are linked with a CAN bus, therefore the communication between them is based on the CANopen protocol. But, in the same way that a computer uses drivers to interact with different peripheral such as a mouse or a keyboard, the robot's computer needs a driver to communicate with the motor motion controllers. And since the motion controller uses the CANopen protocol, the computer's driver must understand this protocol as well.

The objective of this project is to design and develop a CANopen driver that is able to communicate and command a motion controller which is installed in the robot with its main computer. Moreover, this driver should be integrated with the robot's software architecture, thus allowing its different skills to interact with motors when these skills require moving it. The different skills have been developed using the ROS (Robot Operating System) framework. Consequently, the driver has to be integrated into this framework.

The integration of the driver with the ROS framework is demonstrated with the development of a ROS-based application that interacts with the driver. This application consists in a ROS module that enables the control of the robot's wheels with a commercial gaming joypad.

The developed driver can communicate and control any other motion controller as long as it implements the CANopen protocol. This enables the possibility of replacing the motion controller with other one, even from different manufacturer, without the requirement of changing the software.

Resumen

El nuevo robot móvil del grupo Robotics Lab incorpora dos dispositivos de control de motores para ejecutar las órdenes procedentes de su ordenador central. Estas órdenes son transmitidas mediante la comunicación definida por el protocolo CANopen. La ejecución de estas órdenes permite el movimiento de los dos motores instalados en la base del robot. Del mismo modo que un ordenador utiliza drivers para poder interactuar con los diferentes dispositivos periféricos, como por ejemplo un ratón o teclado, el ordenador del robot necesita un driver o controlador que le facilite el envío de órdenes a los dispositivos de control de motores.

En este proyecto se diseña y desarrolla un controlador que establece los medios para que el ordenador del robot pueda manejar, según las especificaciones del protocolo CANopen, los motores de su base. Por tanto, este controlador se llamará controlador CANopen. Por otro lado, las diferentes habilidades del robot se encuentran implementadas en el framework de robots ROS (Robot Operating System) por lo que es necesario la integración del controlador CANopen en dicho framework.

La instalación de este controlador CANopen en el robot permite el manejo de sus motores. En este proyecto además se comprueba el correcto funcionamiento del controlador CANopen mediante el uso de una aplicación facilitada por el framework ROS que permite a un usuario mover el robot mediante un mando inalámbrico.

El controlador CANopen desarrollado puede utilizarse para comunicar cualquier dispositivo de control de motores, que utilice la interfaz CANopen, con el ordenador del robot. Esto permite el uso y configuración de un posible nuevo motor según el tipo funcionamiento que vaya a realizar, es decir, determinando su velocidad máxima, limites de posición, etc. Además, como otra ventaja, el controlador CANopen hace posible la sustitución del modelo hardware del dispositivo de control de motores por otro igual o de diferente fabricante sin necesidad de cambios en el software.

Índice

Índice de ilustraciones

Índice de tablas

1. Introducción

El campo de la robótica está adquiriendo una mayor repercusión en nuestra sociedad, aumentando la funcionalidad y la complejidad de los robots. Uno de los pilares que facilitan esta evolución es la simplificación de sus sistemas de comunicación, los cuales cada vez deben transmitir mayor cantidad de datos. Actualmente existen robots que incorporan dispositivos de distintos fabricantes que utilizan diferentes buses de comunicación. Esto provoca un aumento de la complejidad de la comunicación y cableado del robot.

Aunque el protocolo de comunicación utilizado sea un estándar, el funcionamiento de los diferentes dispositivos que se comunican puede ser específico. Por ejemplo, el protocolo RS232 es un estándar utilizado para comunicar un PC con un modem pero el funcionamiento del modem es específico del fabricante. En el campo de la robótica este funcionamiento específico significa una dependencia del robot hacia el fabricante del dispositivo, por lo que en caso de ser necesaria la sustitución del hardware, se deberá remplazar por un modelo igual si no se desea modificar el software.

Este problema se soluciona utilizando el protocolo CANopen. Este protocolo define tanto las reglas de comunicación, como el funcionamiento de los dispositivos conectados al bus, es decir, define *cómo* y *qué* órdenes deben mandarse para que los dispositivos puedan ejecutarlas. En este caso, la incorporación de nuevos dispositivos al bus o el descarte de aquellos defectuosos puede realizarse sin afectar a los ya instalados. También es posible cambiar un dispositivo por otro de un fabricante diferente, siempre que implementen CANopen, sin necesidad de reconfiguración.

La independencia del hardware y del fabricante que proporciona el protocolo CANopen supone una ventaja en el desarrollo de sistemas robóticos. Esta ventaja se ha aprovechado en el nuevo robot móvil desarrollado por la universidad Carlos III de Madrid. En concreto, se ha decidido comunicar el ordenador del robot con los dispositivos de control de motores, a través de un mismo bus, mediante el protocolo CANopen. Estos dispositivos de control se encargan de ejecutar las órdenes de movimiento procedentes del ordenador del robot para permitir el funcionamiento de los motores de su base. En la Ilustración 1.1 puede observarse el robot móvil Mopi. Este robot cuenta con un motor que mueve las dos ruedas del lado derecho y otro motor que mueve las dos ruedas del lado izquierdo.

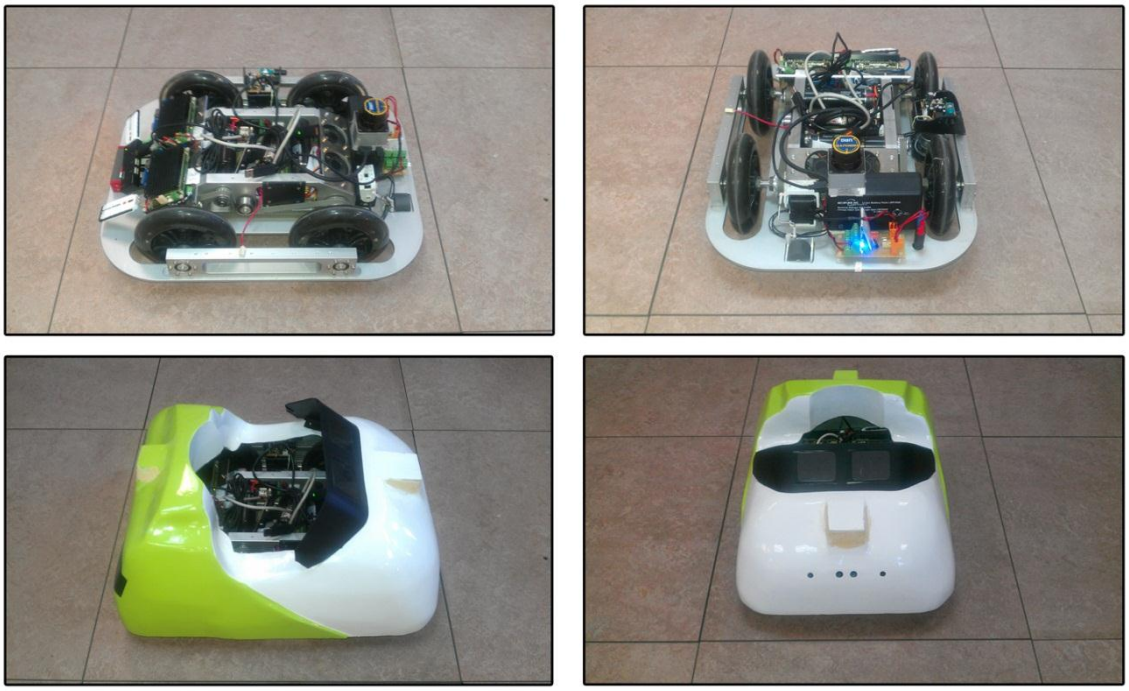


Ilustración 1.1. Robot móvil Mopi.
Fotografías del nuevo robot móvil de la universidad Carlos III.

El ordenador del robot se abstrae del hardware y del protocolo de comunicación utilizado mediante el uso de controladores. Un controlador es un software que permite a un ordenador interactuar con un dispositivo periférico proporcionándole una interfaz de mayor nivel. Por ejemplo, un ordenador necesita un controlador para comunicarse con la impresora. El controlador implementa el protocolo de comunicación utilizado así como las diferentes funcionalidades de la impresora, es decir, si puede imprimir por una o dos caras, en apaisado, en color, etc. El ordenador no necesita conocer que impresora está conectada, simplemente enviará la orden de imprimir un archivo y el controlador se ocupará de resto.

Este proyecto consiste en el diseño y desarrollo de un controlador CANopen para establecer la comunicación del ordenador del robot con los motores de su base mediante el protocolo de comunicación CANopen. Este controlador CANopen permitirá futuras sustituciones de hardware por modelos de otros fabricantes sin necesidad de reconfiguración. En el ordenador del robot se ejecutan diferentes habilidades que definen el funcionamiento del robot. Algunas de estas habilidades deben enviar órdenes a los motores, por ejemplo, una habilidad que permita al robot seguir a la persona que tiene a su lado. Cuando la persona se aleje del robot, la habilidad enviará, a través del controlador CANopen, la orden de mover los motores hacia delante. Las diferentes habilidades se encuentran implementadas dentro del framework de control ROS (Robot Operating System) [1] por lo que el controlador ha sido integrado en dicho framework. El framework de control ROS suministra herramientas y librerías para el desarrollo de aplicaciones software en robots. El concepto de framework puede entenderse con el siguiente ejemplo: imagínese que se desea construir una mesa, para ello serán necesarias diferentes herramientas como un martillo, una sierra, etc. La mesa puede fabricarse en su

casa, donde dispone de una pequeña sierra, o en un taller, donde puede encontrar mejores herramientas como una sierra automática, una pulidora, un nivel, etc. El framework es ese taller que nos aporta una serie de herramientas de calidad para llevar acabo nuestro proyecto. El software se puede implementar sin necesidad de un framework pero siempre se desarrollará con mayor rapidez y eficacia con su uso, es decir, la mesa se fabricará antes y se obtendrá un mejor acabado en un taller.

A las ventajas del protocolo CANopen se deben sumar las suministradas por el framework de control de robots ROS lo que supone una buena combinación para el desarrollo de robots.

1.1 Objetivos

El objetivo de este proyecto es el diseño y desarrollo de un controlador CANopen que permita el manejo de motores. Una vez verificado el funcionamiento de este controlador, se instalará en el nuevo robot móvil Mopi permitiendo que sus diferentes habilidades envíen órdenes a los motores de su base. Para comprobar la correcta instalación, se ha desarrollado una aplicación que permite a un usuario manejar los motores de su base mediante un joypad o mando inalámbrico. Por tanto los objetivos a cumplir en este proyecto son los siguientes:

- Diseño y desarrollo del controlador CANopen.
- Integración del controlador CANopen en el framework de ROS.
- Instalación del controlador CANopen en el robot móvil Mopi para el manejo de los motores de su base.
- Diseño y desarrollo de una aplicación dentro del framework ROS que permita el manejo de la base del robot mediante un mando inalámbrico.

1.2 Solución planteada

La comunicación entre el ordenador del robot y los dispositivos de control de motores se establece según el protocolo de comunicaciones CANopen. Este protocolo establece tanto las comunicaciones como el funcionamiento de diferentes tipos de dispositivos. Todos los dispositivos se conectan al mismo bus permitiendo la reducción de cableado. La comunicación se basa en la transmisión de objetos que determinan las diferentes funcionalidades de los dispositivos. El protocolo CANopen se basa en el protocolo CAN el cual define tanto la capa física como de enlace del modelo OSI [2].

En primer lugar debe seleccionarse el hardware. El modelo del dispositivo de control de motores escogido es el MCDC3006C del fabricante Faulhaber el cual permite la comunicación mediante el protocolo CANopen. El robot consta de dos motores y cada uno de ellos está conectado a un dispositivo MCDC3006C. Estos dispositivos MCDC3006 se conectan al bus de comunicaciones donde también está conectado el ordenador del robot. La abstracción del ordenador del robot tanto del hardware como del protocolo de comunicación utilizado se consigue gracias al desarrollo de un controlador. En este caso debe desarrollarse un controlador que establezca la comunicación definida por el

protocolo CANopen así como las diferentes funcionalidades de los dispositivos de control de motores. El controlador se llamará controlador CANopen.

El desarrollo del protocolo de comunicación CANopen en el controlador CANopen se lleva a cabo mediante la interfaz de programación de aplicaciones (API) de Canfestival [3] que suministra todas las funciones necesarias para el control de una red CANopen. Para permitir el uso del controlador CANopen por parte de las diferentes aplicaciones residentes en el ordenador del robot, se integrará en el framework de programación ROS.

Para comprobar el correcto funcionamiento del controlador y su satisfactoria integración en el robot se ha desarrollado una aplicación, dentro del framework ROS, que permite a un usuario mover el robot mediante mando inalámbrico.

1.3 Estructura del documento

El presente documento se estructura de la siguiente forma:

En el capítulo 2 se hace una breve exposición de diferentes buses de comunicación utilizados en robótica, y en particular del protocolo CANopen el cual ha sido utilizado para el cumplimiento del objetivo de este proyecto.

El capítulo 3 describe tanto el hardware como el software utilizado en el desarrollo del proyecto. El hardware consta del bus de comunicación, un convertidor de puerto USB a CAN que permite la conexión de un PC al bus, así como el dispositivo de control MCDC3006C que ejecutará las órdenes de movimiento de los motores procedentes del ordenador del robot. Posteriormente se presentan los componentes software utilizados, entre ellos, la API de Canfestival que suministra una serie de funciones que permiten el uso del protocolo CANopen. También se hace una breve descripción del framework de control ROS.

El capítulo 4 detalla el desarrollo del proyecto, tanto la arquitectura software diseñada como su implementación. Al final de este apartado se demuestra el funcionamiento del controlador instalado en el robot.

Para finalizar, en el capítulo 5 se presentan las conclusiones adquiridas durante el desarrollo de este proyecto junto con la propuesta de nuevas vías de desarrollo.

2. Buses de comunicación en robótica

Existen diversos buses de comunicación en robótica a través de los cuales se envían los datos. Por ejemplo, si la unidad central de un robot decide que éste debe mover un brazo, se necesita un bus de comunicaciones que envíe las ordenes a todos los motores de dicho brazo. En este capítulo se describirán brevemente los protocolos y buses de comunicación más utilizados en robótica haciendo hincapié en el protocolo CANopen, el cual ha sido utilizado para la realización de este proyecto.

2.1 Protocolo RS232

Hasta hace algunos años, las señales transmitidas eran muy ruidosas, se carecía de filtros y algoritmos robustos, las señales perdían intensidad en los cables y todo esto evitaba la correcta recepción de los datos. Para compensar estas pérdidas se utilizaban señales con voltajes muy altos. Este fue el motivo de la aparición del protocolo de comunicación RS232. Fue desarrollado por lo que hoy se conoce como Electronic Industries Alliance [4]. En los últimos años, el uso de sistemas RS232 ha comenzado a desaparecer dejando paso a otros buses de comunicación como el USB que permite una comunicación fiable a niveles más bajos de tensión.

El bus RS232 permite comunicar un equipo terminal de datos o DTE (Data Terminal Equipment) y un equipo de comunicación de datos o DCE (Data Communication Equipment). Por ejemplo el equipo DTE podría ser un ordenador y el equipo DCE un modem. Se utiliza una señal cuadrada que oscila entre +12V y -12V. El conector utilizado es de tipo DB-25 (de 25 pines), aunque es más frecuente encontrar la versión de 9 pines (DE-9). Su uso se restringe a distancias de hasta 15 metros y velocidades de comunicación de hasta 20 Kilobits/segundo. La comunicación puede ser asíncrona o síncrona en diferentes tipos de canal:

- **Simplex**: En un canal simplex los datos siempre viajarán en una dirección, por ejemplo desde DCE a DTE.
- **Half duplex** : Los datos pueden viajar en ambas direcciones pero sólo durante un determinado periodo de tiempo, posteriormente la línea debe ser conmutada antes que los datos puedan viajar en la otra dirección
- **Full duplex:** Los datos pueden viajar en ambos sentidos simultáneamente.

2.2 Protocolo RS485

El protocolo RS485 es una versión actualizada del protocolo RS232. Mientras que el estándar RS232 permitía la conexión de dos dispositivos, el RS485 soporta la conexión de hasta 32 dispositivos transmisores y 32 receptores con longitudes desde los 10 metros,

trabajando a velocidades de 35 Mbps, hasta los 1200m con velocidades de 100 kbps. El protocolo RS485 define un bus de transmisión de datos, constituido por un par de cables trenzados, que utiliza una comunicación tipo half dúplex, es decir, puesto que hay una sola vía de transmisión sólo puede enviar datos un dispositivo, una vez termine este los demás pueden responder. La aplicación más conocida que utiliza este protocolo es el Profibus. Sus principales características son:

- Interfaz diferencial
- Conexión multipunto
- Alimentación única de +5V
- Hasta 32 estaciones transmisoras y 32 receptoras.
- Velocidad máxima de 100Kbps hasta 1200m y de 10Mbps hasta 12m.
- Longitud máxima de alcance de 1.200 metros (a 100 Kbps)
- Rango de bus de -7V a +12V
- Señales de cómo máximo 6V y de cómo mínimo 200mV.

Las principales diferencias del protocolo RS485 con su predecesor, el RS232, se muestran en la Tabla 2.1.

	RS232	**RS485**
Topología de la red	Punto-punto	Multipunto
Modo de operación	Half duplex / Full duplex	Half duplex
Rango del bus	-12V a +12V	-7V a +12V
Máximo número de emisores	1	32
Máximo número de receptores	1	32
Máxima distancia	15 m	1200 m
Máxima velocidad a 12 m	20 kbs	35 Mbs
Máxima velocidad a 1200 m	---	100 kbs

Tabla 2.1 Características protocolos RS232 y RS485.
Comparación de los protocolos de comunicación RS232 y RS485 utilizados en robótica.

2.3 Protocolo CAN

CAN (Controller Area Network) es un protocolo asíncrono de comunicaciones patentado por la firma alemana Robert Bosch GmbH en 1982, basado en una topología bus para la transmisión de mensajes en entornos distribuidos. El protocolo CAN surgió de la necesidad de encontrar una solución ante el incremento del número de dispositivos electrónicos de los automóviles que suponía un aumento del cableado y complejidad del sistema. El Mercedes Clase E fue el primer coche en incorporar el bus CAN en 1992. Es un protocolo de comunicaciones estandarizado por la International Standardization Organization (ISO) y Society of Automotive Engineers (SAE).

La norma ISO 11898-1 [5] describe el protocolo CAN. La capa física CAN viene descrita en las normas ISO 11898-2 [6] e ISO 11898-3. Para la interfaz MDI (Medium Dependent Interface) que determina el conector físico no existe un estándar pero la CiA (CAN in Automation) [7] en su documento DS-102 recomienda el conector de tipo SUB-D9. Por último la norma ISO 11898-4 determina los tiempos de la comunicación definida en la ISO 11898-1. CAN se corresponde a los niveles de capa física y de enlace del modelo OSI. La relación de las especificaciones ISO determinadas por el protocolo CAN y las diferentes capas del modelo ISO/OSI se pueden observar en la Tabla 2.2.

<table>
<tr><th colspan="2">Modelo ISO/OSI</th><th>Protocolo CAN</th><th>Estándares</th></tr>
<tr><td colspan="2">Capa de aplicación</td><td rowspan="5"></td><td rowspan="5"></td></tr>
<tr><td colspan="2">Capa de presentación</td></tr>
<tr><td colspan="2">Capa de sesión</td></tr>
<tr><td colspan="2">Capa de transporte</td></tr>
<tr><td colspan="2">Capa de red</td></tr>
<tr><td rowspan="2">Capa de enlace</td><td>LLC</td><td rowspan="3">Protocol CAN</td><td rowspan="3">ISO 11898-1</td></tr>
<tr><td>MAC</td></tr>
<tr><td rowspan="4">Capa Física</td><td>PLS</td></tr>
<tr><td>PMA</td><td rowspan="3">CAN Physical Layer</td><td rowspan="2">ISO 11898-2
ISO 11898-3</td></tr>
<tr><td>PMS</td></tr>
<tr><td>MDI</td><td>CiA DS-102</td></tr>
</table>

LLC (Logical Link Control)
MAC (Medium Access Control)
PLS (Physical Layer Signalling)
PMA (Physical Medium Attachement)
PMS (Physical Medium Specification)
MDI (Medium Dependent Interfave)

Tabla 2.2 Especificaciones modelo ISO/OSI para CAN.
CAN se corresponde a los niveles de capa física y de enlace del modelo OSI.

2.3.1 Características del protocolo CAN

Como se menciona anteriormente, CAN se trata de un protocolo estandarizado que define un único bus de comunicación donde son conectados los diferentes dispositivos. Al ser una red multiplexada, reduce considerablemente el cableado y elimina las conexiones punto a punto, excepto en los enganches.

CAN se basa en un protocolo de transporte orientado a mensajes. Se definen el contenido del mensaje en lugar de las estaciones (nodos) y sus direcciones lo cual aporta un alto grado de flexibilidad de configuración del sistema. Es fácil agregar estaciones a una red existente CAN sin realizar ninguna modificación de hardware o software para las estaciones actuales. Por ejemplo puede pensarse en un nuevo modelo de coche que incorpore nuevas funcionalidades como sensores de proximidad, utilizados a la hora de aparcar. En este caso dichos sensores podrían será instalados en el bus CAN del modelo de coche antiguo asignándoles sus propios identificadores de mensajes así como a la unidad

de control sin tener que realizar ningún cambio en otros nodos de la red como por ejemplo el climatizador o el ordenador de a bordo. Este concepto modular permite un mantenimiento simple y modernización de la red. Un esquemático de la red CAN puede observarse en la Ilustración 2.1:

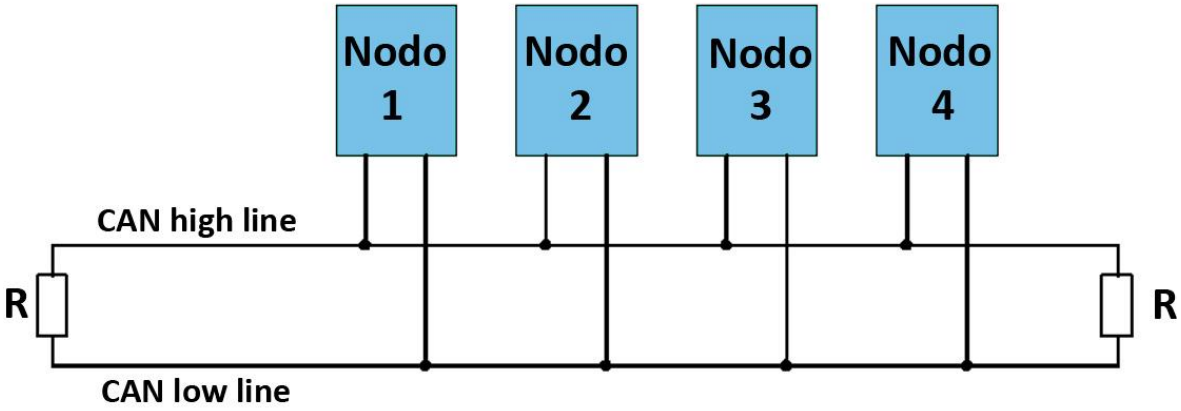


Ilustración 2.1: Esquema de una red CAN.
El bus de comunicación está formado por dos cables trenzados con elementos terminadores en sus extremos. A través del bus viajan las tramas de datos que son enviadas/recibidas por los dispositivos conectados. Cada dispositivo conectado al bus representa un nodo.

Las principales características del protocolo CAN son:

- Un único bus de transmisión.
- Velocidad de comunicación de hasta 1 Mbit/s. La velocidad de transmisión puede ser diferente en distintos sistemas. En un sistema la velocidad debe de ser uniforme.
- Se emplean dos cables por los cuales viajan dos señales exactamente iguales en amplitud y frecuencia pero completamente inversas en voltaje. Los módulos con estos dos pulsos identifican el mensaje, pero también tiene opciones de mantener la red activa aunque falle uno de los cables de comunicación.
- Valores de Bus (Dominante o recesivo). Los nodos conectados al bus interpretan dos niveles lógicos denominados:
  - Dominante: la tensión diferencial (CAN_H - CAN_L) es del orden de 2.0 V con CAN_H = 3.5V y CAN_L = 1.5V (nominales).
  - Recesivo: la tensión diferencial (CAN_H - CAN_L) es del orden de 0V con CAN_H = CAN_L = 2.5V (nominales).
- Los mensajes CAN poseen un formato fijo que puede tener diferentes longitudes según el tipo de trama.

- Cada mensaje tiene un identificador de mensaje, que es único en toda la red. El contenido de un mensaje se especifica por dicho identificador, el cual no indica el destino, sino describe el significado del mensaje. Además indica la prioridad del mensaje, lo cual es importante en el momento en que varias estaciones (nodos) compiten por el acceso al bus (arbitraje de bus).

- La petición de datos remotos se realiza enviando primero una trama remota de petición de trama que posee una determinado identificador y esta será contestada con otra definida con el mismo identificador.

- Flexibilidad de la configuración.

- Multimaestro:

  - Si el bus está libre, cualquier nodo puede comenzar a transmitir un mensaje.
  - Cuando dos nodos comienzan a transmitir simultáneamente el conflicto de acceso al bus es resuelto por arbitraje utilizando el identificador que determina que mensaje es más prioritario. El mecanismo de arbitraje garantiza que ni la información ni el tiempo se pierdan.
  - Cuando una trama de datos y una trama remota se inician al mismo tiempo prevalece la primera.
  - Durante el arbitraje todos los transmisores comparan el nivel del bit transmitido con el nivel del bus: Si los niveles son iguales, la unidad puede enviar. Si son distintos, la unidad pierde el arbitraje y debe retirarse sin enviar otro bit.

- Detección de error y señalización: En todos los nodos CAN se implementan medidas especiales para la detección de errores, señalización y auto-chequeo.

2.4 Protocolo CANopen

CANopen cubre tanto un protocolo de comunicación como una serie de especificaciones que deben cumplir los dispositivos en sistemas embebidos utilizados en automática. Consiste en un esquema de direccionamiento, protocolos de comunicación y una capa de aplicación definida por el perfil del dispositivo. Los protocolos de comunicación proporcionan un control de la red, monitoreo de los nodos y su comunicación así como la capa de transporte de mensajes. Este protocolo está basado en CAN el cual implementa las capas de enlace y física como se ha explicado en el apartado anterior. CANopen se utiliza en varios campos como el control de maquinaria, dispositivos médicos, trenes, automoción etc.

CANopen constituye la capa de aplicación del modelo de comunicación OSI. Este protocolo está basado en el protocolo CAN el cual, como se comento anteriormente, constituyen las capas de enlace y física del modelo OSI. La gran ventaja que aporta CANopen respecto de CAN es su comunicación basada en objetos, la cual permiten la comunicación de datos, detección de errores, control de la red, etc. de forma sencilla gracias a la abstracción de las tramas CAN. Cada objeto supone una funcionalidad diferente y todos los objetos de un dispositivo se almacenan en lo que se denomina diccionario de objetos que posteriormente se explicará en el apartado 2.4.3.

En la Ilustración 2.2 puede observarse como se realiza la comunicación mediante objetos del protocolo CANopen. El dispositivo emisor envía el objeto, por ejemplo indicando que velocidad debe alcanzar el motor, el dispositivo receptor guarda el valor del objeto enviado en su diccionario de objetos y ejecuta dicha orden.

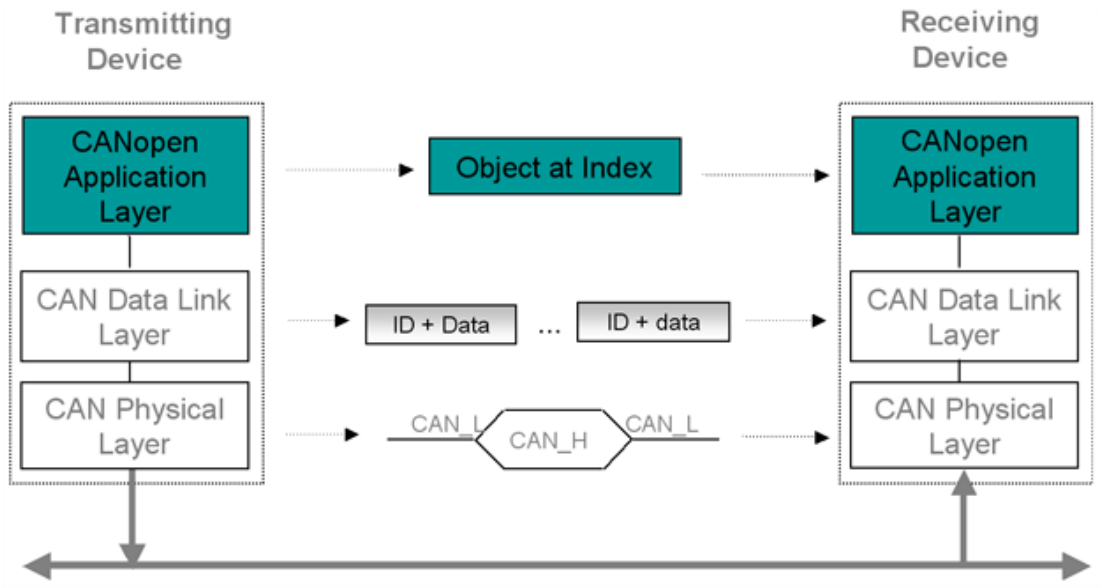


Ilustración 2.2: Esquema de comunicación CANopen.
Las capas Data Link y Phisical están definidas mediante el protocolo CAN mientras que la capa Application está definida por CANopen y permite la comunicación mediante objetos los cuales corresponden a diferentes funcionalidades de los dispositivos. Ilustración obtenida de CiA [7].

El protocolo de comunicación CANopen se encuentra definido en el documento DSP-301 de la CiA [8]. La CiA también especifica las funcionalidades de los diferentes dispositivos. En concreto, las funcionalidades de los dispositivos de control de motores, utilizados en este proyecto, se encuentran en el documento DSP-402 [9].

2.4.1 Modelos de comunicación.

Los diferentes modelos de comunicación permiten la transmisión asíncrona y síncrona de mensajes. La transmisión síncrona de mensajes viene determinada por la previa definición de los objetos de comunicación (mensajes de sincronización *SYNC* y *Time stam*p) en los que se establecen los tiempos y frecuencias de envío. Los mensajes asíncronos pueden ser transmitidos en cualquier momento. Existen tres tipos de modelos:

Maestro/Esclavo: Un solo maestro en toda la red y los demás nodos son considerados esclavos. Puede existir comunicación con o sin confirmación. En la Ilustración 2.3 e Ilustración 2.4 pueden observarse ambos tipos de comunicación.

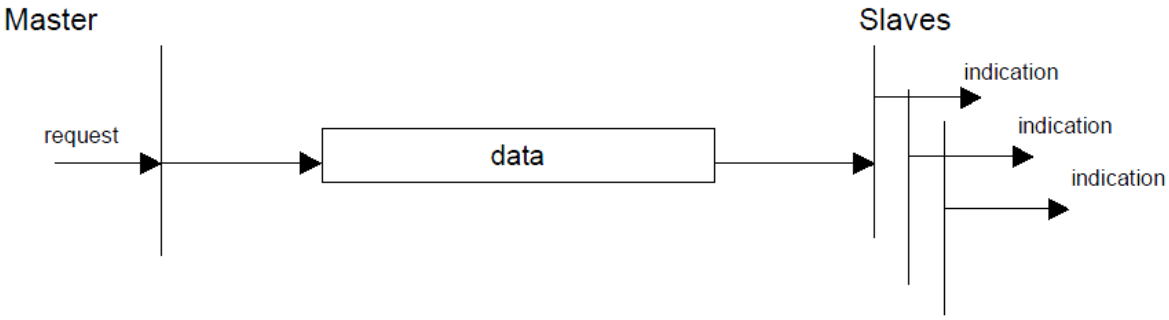


Ilustración 2.3: Comunicación Maestro/Esclavos sin confirmación.
El nodo maestro de la red CANopen envía un objeto a uno o varios nodos esclavos. No se obtiene respuesta por parte de los esclavos. Ilustración obtenida de CiA **[8]**.

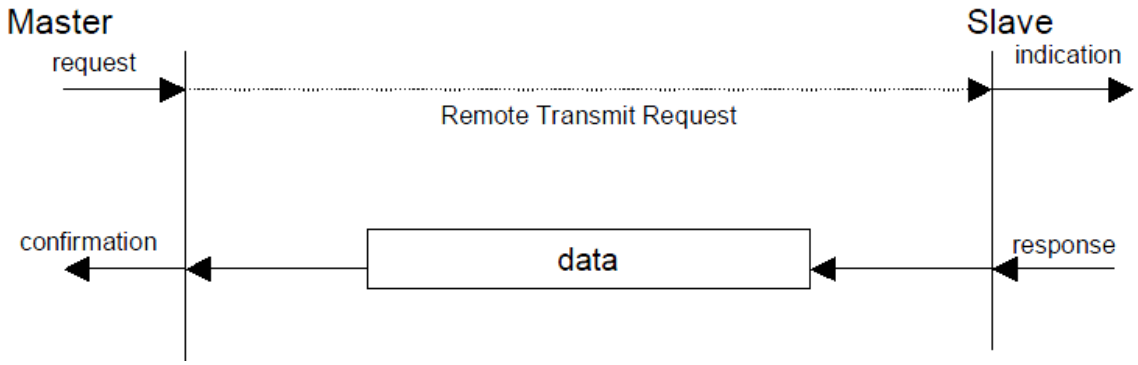


Ilustración 2.4: Comunicación Maestro/Esclavos con confirmación.
El nodo maestro envía una trama de petición de datos a un solo esclavo y es contestado por el este con el envío de los datos solicitados. Ilustración obtenida de CiA [8].

Cliente/Servidor: La comunicación se lleva a cabo entre un único cliente y un único servidor. El cliente realiza una petición de escritura o lectura (upload/download) para que el servidor realice una determinada tarea. Una vez terminada dicha tarea el servidor contesta la petición. Este tipo de comunicación es el más usado, por ejemplo el cliente es la unidad de control del robot y solita la velocidad del motor al dispositivo de control de motores. El dispositivo envía al cliente la velocidad actual del motor. Ver Ilustración 2.5.

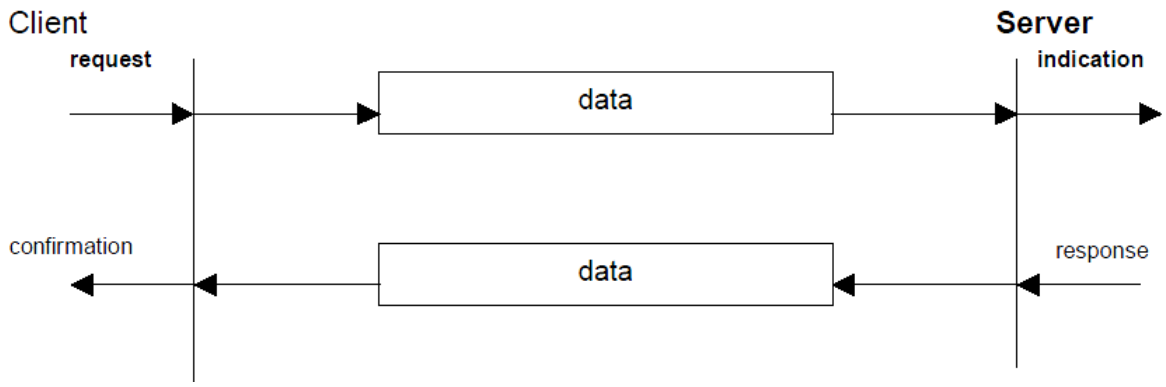


Ilustración 2.5: Comunicación Cliente/Servidor.
El cliente envía una trama de escritura/lectura en los datos del servido y es confirmada por este. Ilustración obtenida de CiA [8].

Productor/Consumidor: Dicha comunicación está formada por un productor y ninguno o más consumidores. Puede ser una comunicación tipo:

- Pull: Con confirmación por parte del productor a la petición realizada por el consumidor. Véase Ilustración 2.6.

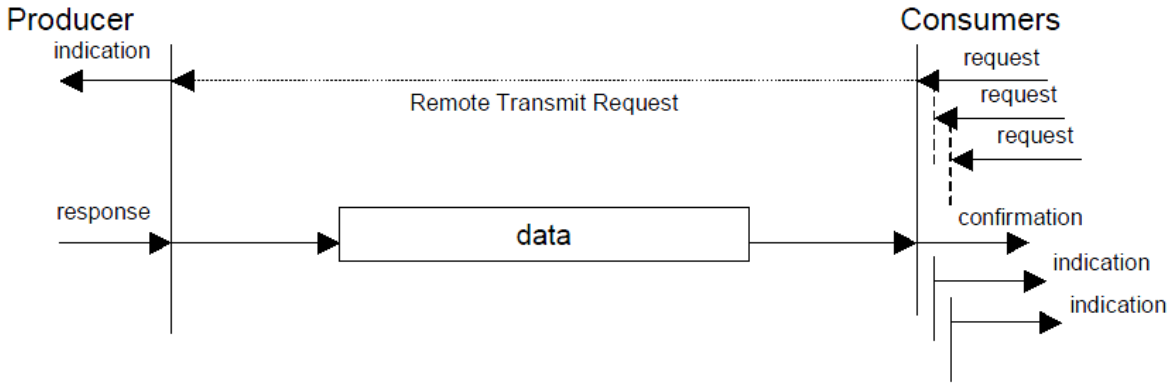


Ilustración 2.6: Comunicación Pull Productor/Consumidor.
Los consumidores envían tramas de petición al productor el cual les contesta con los datos utilizados. Ilustración obtenida de CiA **[8]**.

- Push: Las peticiones son realizadas por el productor. Sin confirmación. Véase Ilustración 2.7.

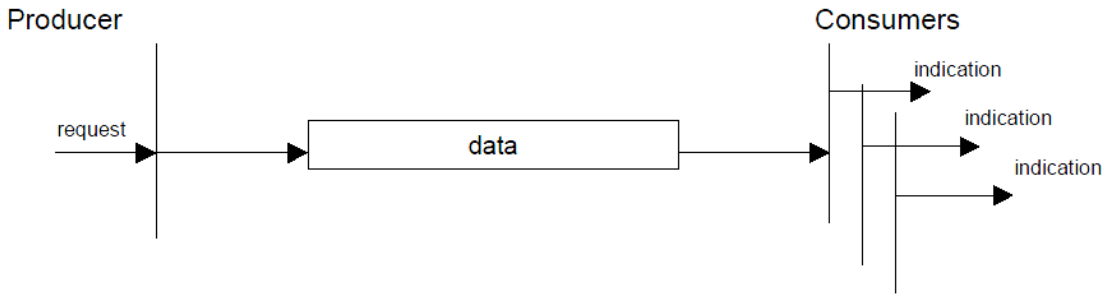


Ilustración 2.7: Comunicación Push Productor/Consumidor.
El productor envía los objetos a ninguno o más consumidores. Ilustración obtenida de CiA **[8]**.

2.4.2 Modelo de los dispositivos

Los diferentes dispositivos conectados a una red CANopen constan de tres bloques como se muestra en la Ilustración 2.8.

- **Comunicación**: Permite la comunicación entre objetos y garantiza la correcta funcionalidad en el transporte de datos mediante las capas inferiores.

- **Diccionario de objetos:** Es la agrupación de los diferentes datos los cuales influyen en el comportamiento de las aplicaciones, comunicación entre objetos y la máquina de estados del dispositivo.

- **Aplicación:** En este bloque se establece la funcionalidad del dispositivo.

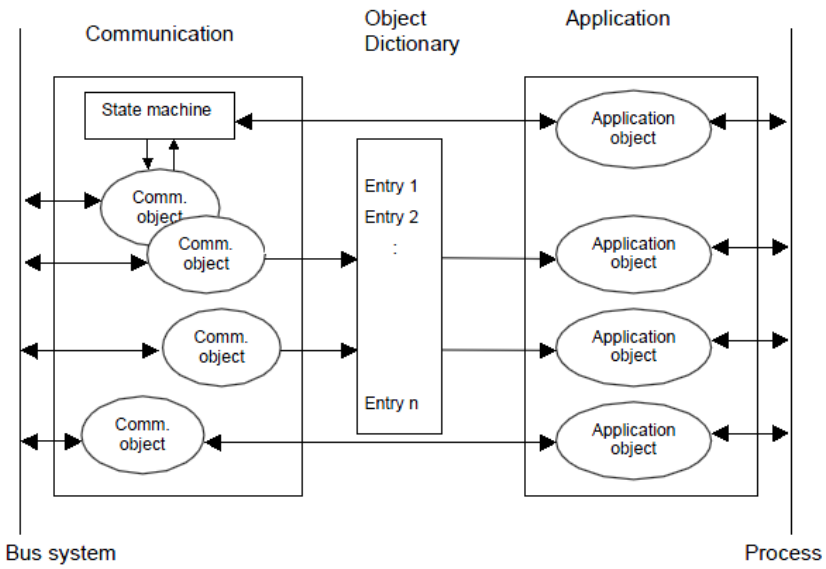


Ilustración 2.8: Modelo de los Dispositivos.
Todos los dispositivos de una red CANopen están constituidos por tres bloques: El bloque de comunicación, el diccionario de objetos y el bloque de aplicación. Ilustración obtenida de CiA **[8]**.

2.4.3 Diccionario de objetos de CANopen

El concepto central de CANopen es el diccionario de objetos (OD, *Device Object Dictionary*). Este es un grupo ordenado de objetos que describe de forma estandarizada la funcionalidad de cada dispositivo y permite su configuración mediante mensajes (SDO***: Service Data Objects***) a través del propio bus. Por ejemplo, un dispositivo controlador de motores incorpora un diccionario de objetos entre los que se encuentran el objeto donde se almacena la velocidad que se quiere alcanzar, otro objeto para almacenar el valor de la velocidad actual del motor, otro contiene la velocidad máxima, etc.

Los objetos se encuentran agrupados en el diccionario según su funcionalidad. Para ello, los objetos son organizados mediante una dirección de almacenamiento descrita por un Índice de 16 bits y un subíndice de 8 bits. Cada índice junto con el subíndice se denomina entrada del diccionario de objetos y dependiendo del objeto tienen distintos tipos de datos. Un objeto puede estar compuesto de varias entradas. Por ejemplo el objeto de la Ilustración 2.9 define los límites de posición para los motores, según las especificaciones que se encuentran en el documento DSP-402 de la CiA [9] el objeto tiene el índice 0x607D. En el subíndice 0x00 el dato que se almacena es de tipo unsigned8 e indica el número elementos que constituyen dicho objeto, en el subíndice 0x01 se almacena el límite de posición menor de tipo integer32 y en el subíndice 0x02 se almacena el límite de posición mayor también de tipo integer32.

Software Position Limit

Index	Subindex	Name	Type	Attrb.	Default value	Meaning
0x607D	0	number of entries	Unsigned8	ro	2	Number of object entries
	1	min position limit	Integer32	rw	see spec.	Lower positioning range limit
	2	max position limit	Integer32	rw	see spec.	Upper positioning range limit

Ilustración 2.9. Estructura de un objeto.
El objeto Software Position Limit, que define los límites de posición, está compuesto por dos subíndices que contienen el límite de posición alto y el bajo. Los tipos de datos están definidos en las especificaciones de la CiA. Ilustración obtenida de Faulhaber **[9]**.

El Maestro puede acceder a los diccionarios de los objetos de los esclavos y modificar los valores de sus objetos sólo si la entrada al diccionario de objetos del dispositivo está permitida. El maestro puede tener acceso de lectura o de escritura/lectura a las entradas de los dispositivos. Cada nodo de la red tiene su propio diccionario de objetos, que contiene todo los parámetros que describen el dispositivo y su comportamiento en la red.

Existen diferentes tipos de objetos que definen:

- **Tipos de datos estándar:** Integer8, unsigned32, reales, booleanos …

- **Tipos específicos de datos:** Estos objetos definen tipos de datos específicos del fabricante para determinado dispositivo.

- **Perfiles de comunicación:** donde están descritos todos los parámetros relacionados con las comunicaciones de una red CAN.

- **Parámetros específicos del fabricante:** El fabricante puede definir objetos para mejorar la funcionalidad de sus dispositivos que no se encuentre en los perfiles de la CiA [7].

- **Perfiles de dispositivo:** donde se definen los objetos propios de un tipo de dispositivo, por ejemplo en este proyecto englobarían los objetos que almacenan la velocidad del motor, la corriente, los que configurar la velocidad máxima, etc.

Los objetos son agrupados en el diccionario según la función que desempeñen como se puede observar en Tabla 2.3. Por ejemplo, el grupo *Standardized Device Profile* reúne todos los objetos que determinan las funcionalidades de un determinado dispositivo, es decir, si se trata de una tarjeta de adquisición de datos se almacenan los objetos relativos a las entradas y salidas digitales, entradas y salidas analógicas, etc. Por el contrario si es un dispositivo de control de motores en este grupo se almacenan los objetos que determinan la velocidad del motor, la corriente consumida, la posición, etc. En el caso del objeto *Software Position Limit*, encargado de establecer los límites de posición de los motores, se encuentra dentro del grupo *Standardized Device Profil* puesto que se trata de un objeto propio de los dispositivos de control como se recoge en el documento DS402 de la CiA [9].

Rango de Índices	Nombre del grupo de objetos
0x0001-0x0FFF	Data Types
0x1000-0x1029	Communication Parameters
0x1200-0x12FF	SDO Parameters
0x1400-0x15FF	Receive PDO Paramameters
0x1600-0x17FF	Receive PDO Mapping
0x1800-0x19FF	Transmit PDO Paramameters
0x1A00-0x1BFF	Transmit PDO Mapping
0x2000-0x9FFF	Manufacturer Specific
0x6000-0x9FFF	Standardized Device Profile

Tabla 2.3 Composición del diccionario de objetos.
Los objetos son agrupados según el tipo de función que realicen.

En la Ilustración 2.10 se muestra un ejemplo de una sección de un diccionario de objetos. En concreto se observar una serie de objetos pertenecientes al grupo *Communication Parameters* que almacenan información sobre la comunicación. En concreto los objetos que aparecen son:

- Device Type Information: Está constituido por una entrada del diccionario con índice 0x1000h y subíndice 0x00h. Este objeto almacena información del tipo de dispositivo al que pertenece, es decir, si se trata de un dispositivo de control, una tarjeta de adquisición de datos, etc.

- Error Register: Este objeto se encuentra en la entrada del diccionario con índice 0x1001h y subíndice 0x00h. En este objeto se almacenan los erros de comunicación ocurridos en la red CANopen.

- Heartbeat Time: Al igual que los objetos anteriores consta de solo una entrada con índice 0x1017h y subíndice 0x00. Establece el tiempo mensajes del tipo HeartBeat.

- Identity Object: Almacena información sobre el dispositivo en la red. En este caso este objeto está constituido por cinco entradas:

    - Número de entradas: Se establece el número de entradas con información además de ésta. En esta entrada se encuentra el valor 4. Entrada con índice 0x1018 y subíndice 0x00
    - VendorID. Entrada con índice 0x1018 y subíndice 0x01
    - Product Code. Entrada con índice 0x1018 y subíndice 0x02
    - Revision Number. Entrada con índice 0x1018 y subíndice 0x03
    - Serial Number. Entrada con índice 0x1018 y subíndice 0x04

Index	SubIdx	Type	Description
1000h	0	UNSIGNED32	Device Type Information
1001h	0	UNSIGNED8	Error Register
1017h	0	UNSIGNED16	Heartbeat Time
1018h			Identity Object
	0	UNSIGNED8	= 4 (Number of sub-index entries)
	1	UNSIGNED32	Vendor ID
	2	UNSIGNED32	Product Code
	3	UNSIGNED32	Revision Number
	4	UNSIGNED32	Serial Number

Ilustración 2.10: Diccionario de Objetos.
Se pueden observar cuatro objetos diferentes: De los cuatro, los tres primeros están formados por una sola entrada del diccionario. El ultimo objeto, Identity Object, consta de cinco entradas.

2.4.4 Comunicación entre objetos

El modelo de comunicaciones de CANopen define cuatro tipos de objetos (objetos de comunicación):

- **Objetos administrativos:** son mensajes administrativos que permiten la configuración de las distintas capas de la red así como la inicialización, configuración y supervisión de la misma.

- **Service Data Objects (SDO):** son objetos utilizados para leer y escribir cualquiera de las entradas del diccionario de objetos de un dispositivo. Corresponden a

mensajes CAN de baja prioridad. Explicados más extensamente en el apartado 2.4.5.

- **Process Data Objects (PDO):** los PDO establecen los mensajes de proceso utilizados para el intercambio de datos de proceso, es decir, datos de tiempo real. Por este motivo, típicamente corresponden a mensajes CAN de alta prioridad. Posteriormente se explicarán con mayor profundidad en el apartado 2.4.6.

- **Objetos predefinidos:** estos objetos definen los mensajes de sincronización (SYNC), de emergencia y *Time Stamp*. Estos objetos permiten la sincronización de los dispositivos (mensajes SYNC) y generar notificaciones de emergencia.

2.4.4.1 Identificadores de mensaje

El protocolo CANopen establece un identificador para cada mensaje, siendo único para los mensajes *NMT* (Network Management), *SYNC* y *Time stamp*. Para reducir el tiempo de configuración de una red CANopen existe un esquema predefinido de asignación de identificadores de 11 bits para los mensajes, denominados *Predefined Connection Set*. Este esquema define los identificadores para un mensaje de emergencia por nodo, mensajes de sincronización, mensaje time stamp, un SDO (ocupando dos identificadores), mensajes NMT y hasta cuatro PDOs de transmisión y cuatro de recepción por dispositivo.

En la Ilustración 2.11 se puede observar el identificador de 11 bits, denominado COBID (Comunication Objects Identifier). El COBID se divide en dos partes:

- 4 bits para el código de función. Pueden observarse los diferentes códigos en la Tabla 2.4.
- 7 bits para el identificador de nodo (Node-ID).

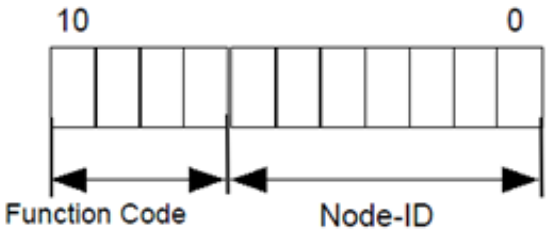


Ilustración 2.11: Estructura del identificador de mensajes CANOpen (COBID).
Cada mensaje posee un identificador único denominado COBID. Consta del código de función del mensaje y el identificador del nodo que lo envía. Ilustración obtenida de CiA **[8]**.

La distribución de los identificadores se corresponde con una estructura del tipo maestro-esclavo. El maestro conoce los identificadores ID de todos los esclavos conectados (máximo 127) mientras que los esclavos no pueden comunicarse entre sí porque no conocen los identificadores de los demás. En las Tabla 2.4 se puede observar la distribución general de los identificadores por defecto según el tipo de mensaje junto con

el índice del diccionario de objetos donde se encuentra el objeto. Con menor COBID se encuentran los mensajes de mayor prioridad.

Objeto	Código de Función	COBID	Índice
NMT	0000	0x00	---
SYNC	0001	0x80	0x1005h, 0x1006h, 0x1007h
TYME STAMP	0010	0x100	0x1012, 0x1013
EMERGENCY	0001	0x81-0xFF	0x1014, 0x1015
PDO1 (tx)	0011	0x181-0x1FF	0x1800
PDO1(rx)	0100	0x201-0x27F	0x1400
PDO2 (tx)	0101	0x281-0x2FF	0x1801
PDO2(rx)	0110	0x301-0x37F	0x1401
PDO3 (tx)	0111	0x381-0x3FF	0x1802
PDO3(rx)	1000	0x401-0x47F	0x1402
PDO4 (tx)	1001	0x481-0x4FF	0x1803
PDO4(rx)	1010	0x501-0x57F	0x1403
SDO (tx)	1011	0x581-0x5FF	0x1200
SDO(rx)	1100	0x601-0x67F	0x1200
NMT Control de error	1110	0x701-0x77F	0x1016, 0x1017

Tabla 2.4: COBIDs de los diferentes tipos de mensajes en la red CANopen.
Se muestra una configuración de COBIDs por defecto suministrada por CANopen para facilitar la puesta en marcha de la red. Tabla obtenida de CiA **[8]**.

Como se puede observar en la Tabla 2.4, los mensajes más prioritarios son los definidos por el objeto NMT Module Control cuyo COBID es 0x00h. El máximo número de nodos en una red CANopen es 127 por lo que cada tipo de mensaje tiene asignado un rango de COBID. Por ejemplo el rango para los PDO1 transmit (TxPDO1) comienza con el valor COBID= 0x180 + NodeID, es decir, desde el COBID=0x181h para el nodo con NodeID=0x01 hasta el COBID=0x1FF para el nodo con identificador NodeID=0x7F=127d. Los mensajes *NMT*, *SYNC* y *Time Stamp* con un único COBID son recibidos por todos los nodos.

2.4.5 Service Data Objects (SDO)

Los mensajes SDO permiten la lectura y escritura de las entradas de los diccionarios de objetos de los dispositivos para su configuración y transferencia. Esta comunicación se puede llevar a cabo mediante los modelos de comunicación cliente-servidor o punto-a-punto. Para todos los tipos de transmisión, el cliente es aquel que inicia la transferencia de datos de configuración siendo el servidor el dispositivo cuyo diccionario está siendo accedido. Los dispositivos deben soportar al menos un tipo de mensaje SDO. Tanto el servidor como el cliente pueden abortar la transferencia de los mensajes SDO. En comparación con los PDOs, son mensajes de baja prioridad.

El cliente puede controlar mediante el índice y el subíndice la entrada del diccionario de objetos donde serán escritos o leídos los datos. Un intercambio de datos mediante

mensajes SDO consta del envío dos tramas con diferentes identificadores, una trama desde el cliente al servidor y otra, con diferente COBID, desde el servidor hasta el cliente. Es importante tener en cuenta que en CANopen los parámetros de más de un *byte* se envían primero el byte menos significativo o LSB (Less Significative Bit). Por ejemplo el cliente, la unidad de control del robot, enviará una trama SDO con COBID= 0x601 al servidor, una tarjeta de adquisición de datos con número de identificación 0x01, para conocer el valor de la primera entrada digital que está almacenada en la entrada del diccionario de objetos con índice 0x6000 y subíndice 0x00. El servidor contesta, mediante un mensaje SDO con COBID=0x581, el valor de dicha entrada.

Los mensajes SDO son transferidos como una secuencia de segmentos denominada **transferencia por segmentos** en la que debido al gran número de datos a transmitir deben enviarse varias tramas que contienen hasta siete bytes de datos, lo que se conoce como segmento. Existe una fase de inicialización donde el cliente y el servidor se preparan para transmitir los segmentos indicando el número de ellos que se desea enviar, posteriormente comenzara la fase de transferencia donde se transmitirá el número de segmentos acordado. Si el número de datos a transmitir es inferior a cuatro bytes es posible realizarlo durante la fase de inicialización lo que se conoce como **transferencia expedita**.

Opcionalmente un SDO puede ser transferido como un conjunto de bloques donde cada uno es una secuencia de hasta 127 segmentos que contienen el número de secuencia y los datos. Del mismo modo que en la transferencia expedita existe una fase de inicialización donde el cliente y el servidor se preparan para transmitir los bloques y negociar el número de segmentos por bloque. Posteriormente a la transmisión de bloques existe una fase de finalización donde el cliente y el servidor pueden opcionalmente verificar si los datos transferidos son correctos mediante la comparación de los checksum. Este proceso se denomina **transferencia de Bloques.**

En el anexo E se hace hincapié solamente en la transferencia expedita puesto que es el tipo de transferencia utilizada en este proyecto. Si se desea saber más acerca de la transferencia por segmentos o bloques pueden consultarse las especificaciones definidas por la CiA [8].

2.4.6 Process Data Objects (PDO)

Los dispositivos de una red CANopen también pueden comunicarse mediante mensajes PDO (Process Data Objects). Este tipo de mensajes permite intercambiar datos en tiempo real. Utiliza el modelo de comunicaciones productor-consumidor. Los datos se transmiten desde un productor a varios consumidores. Estos mensajes ofrecen un servicio de transferencia de datos sin confirmación.

Los mensajes PDO de un nodo o dispositivo pueden dividirse en dos categorías. Los mensajes de transmisión o TxPDO son aquellos mensajes con información que el nodo transmite (por ejemplo un dispositivo de control de motores transmite objeto Statusword permite observar si el motor está habilitado o no). Por otro lado, los mensajes de

recepción o RxPDO son los mensajes que el nodo escucha (por ejemplo un dispositivo de control de motores recibe el objeto Controlword que permite habilitar o deshabilitar el motor).

El contenido de un PDO está definido tan sólo por su identificador. Tanto el emisor como el receptor deben conocerlo para poder interpretar su estructura interna. Cada PDO se describe mediante dos objetos del diccionario:

- **PDO Communication Parameter**: contiene el COBID (identificador de mensaje) que utiliza el PDO, el tipo de transmisión, tiempo de inhibición y temporizador.

- **PDO Mapping Parameter:** contiene una lista de objetos del diccionario de objetos contenidos en el mensaje PDO.

CANopen define varios mecanismos de comunicación para la transmisión de PDOs:

- **Transmisión asíncrona:** El envío de los mensajes PDOs se produce mediante la aparición de un evento determinado como por ejemplo la recepción de una trama remota solicitando datos.

  - **Eventos (Event Driven):** la transmisión de un mensaje es causada por la aparición de un evento específico definido en el perfil del dispositivo.

  - **Temporizador (Timer Driven):** existe un temporizador que cada cierto tiempo cause la transmisión.

  - **Solicitud remota (Remotely requested):** la transmisión asincrónica de mensajes PDO puede comenzar al recibir una solicitud remota (trama RTR) enviada por otro dispositivo.

- **Transmisión síncrona:** La transmisión síncrona de mensajes PDO es disparada por la expiración de un período de transmisión, sincronizado mediante la recepción de objetos SYNC. Es decir, cada vez que llega un mensaje SYNC, se abre una ventana de transmisión sincrónica. Los PDOs sincrónicos deben ser enviados dentro de esa ventana. Se distinguen dos modos dentro de este tipo de transmisión:

  - **Modo cíclico**: son mensajes que se transmiten dentro de la ventana abierta por el objeto SYNC. No se transmiten en todas las ventanas sino con cierta periodicidad, especificada por el campo Transmission Type del Communication Parameter correspondiente.

  - **Modo acíclico**: son mensajes que se transmiten a partir de un evento de la aplicación. Se transmiten dentro de la ventana pero no de forma periódica.

La Ilustración 2.12 muestra los modos de comunicación asíncrona y síncrona anteriormente descritos.

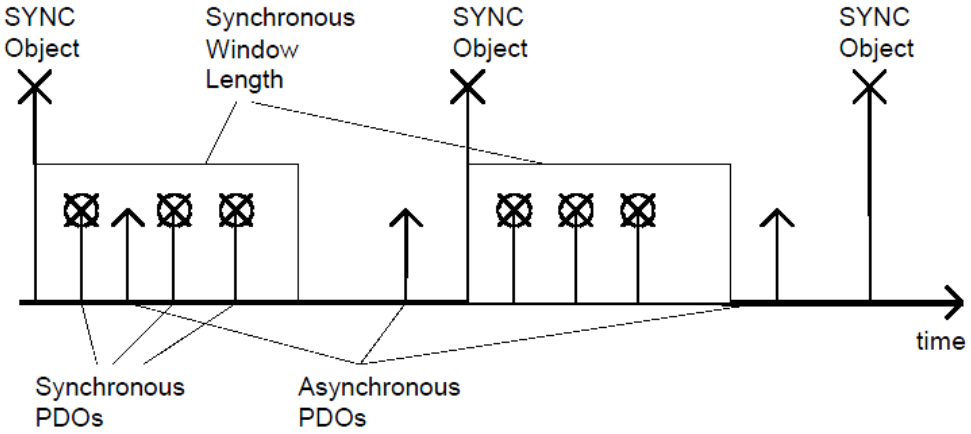


Ilustración 2.12: Tipos de transmisiones de PDOs.
El envío de mensajes PDOs puede ser asíncrono, enviados cuando se reciba un evento, o síncrono mediante el envío de mensajes SYNC o sincronización enviados cíclicamente. Ilustración obtenida de CiA **[8]**.

Un PDO puede tener asignado un tiempo de inhibición que define el tiempo mínimo que debe pasar entre dos transmisiones consecutivas del mismo PDO. Forma parte del *Communication Parameter*. Está definido como un entero de 16 bits en unidades de 100 microsegundos.

- **Write PDO**

  Se utiliza el modelo de comunicación push. Pueden o no existir consumidores y un solo un productor por mensaje PDO. Mediante este servicio el productor envía las direcciones de los objetos contenidos en el mensaje PDO al consumidor. No existe confirmación. Ver Ilustración 2.13.

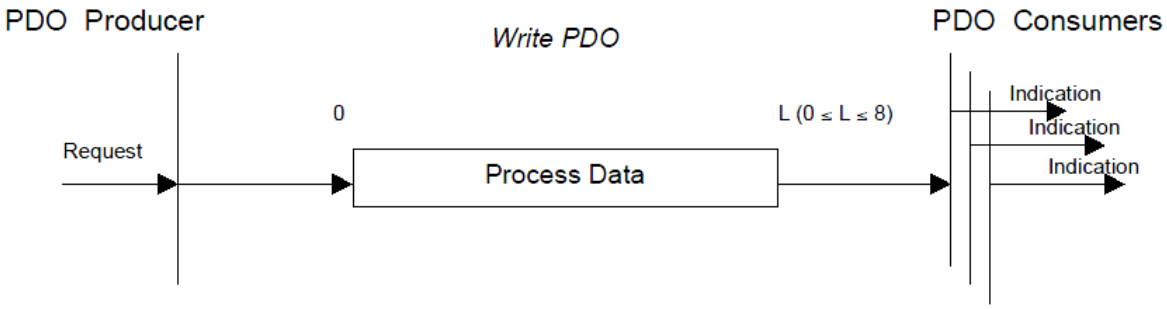


Ilustración 2.13. Protocolo Escritura PDO Push.
La escritura mediante mensajes PDO utiliza el modelo push productor-consumidores. Ilustración obtenida de CiA **[8]**.

En caso de que se exceda el número de bytes indicados por el tamaño del PDO *mapping* no se usaran por el consumidor, utilizándose solo aquellos que este por debajo de dicho valor. Si se reciben menos bytes de los que se esperaban no son procesados y se envía el mensaje de error.

- **Read PDO**

  Se utiliza el modelo de comunicación pull. Deben existir uno o más consumidores y un solo un productor por mensaje PDO. Mediante este servicio los productores envían los datos que se encuentran en las direcciones de los objetos previamente solicitadas por el consumidor mediante una trama remota. Existe confirmación. Véase la Ilustración 2.14.

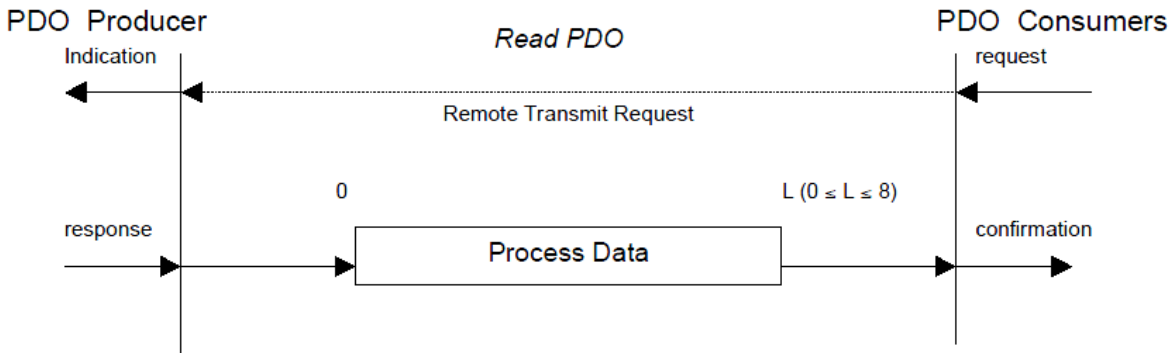


Ilustración 2.14. Protocolo Lectura PDO pull.
La lectura mediante mensajes PDO utiliza el modelo push productor-consumidores. Ilustración obtenida de CiA **[8].**

Al igual que para el protocolo de escritura, en caso de que se exceda el número de bytes indicados por el tamaño del PDO *mapping* no se usaran por el consumidor, utilizándose solo aquellos que este por debajo de dicho valor. Si se reciben menos bytes de los que se esperaban no son procesados y se envía el mensaje de error.

2.4.7 Mensaje de emergencia

Estos mensajes se envían cuando ocurre un error interno en un dispositivo. Se transmiten al resto de dispositivos con la mayor prioridad. Pueden usarse como interrupciones o notificaciones de alertas. Estos mensajes siguen el modelo de comunicación productor/consumidor. No existe confirmación.

Un mensaje de emergencia tiene 8 *bytes*. Su estructura se muestra en la Tabla 2.5.

Byte	0	1	2	3	4	5	6	7
Contenido	Emergency Error Code		Error register	Manufacturer specific Error Field				

Tabla 2.5. Estructura de un mensaje de emergencia en CANopen.
Muestra el contenido de un mensaje de emergencia. Tabla obtenida de CiA **[8].**

- **Emergency Error Code** (2 *bytes*): código del error que causó la generación del mensaje de emergencia. Se trata de fallos internos de los dispositivos por lo cual, los errores se relacionan con fallos de tensión, de corriente, del software del dispositivo, del adaptador del bus CAN, etc. La Tabla 2.6 nos muestra los códigos correspondientes en hexadecimal:

Error Code (hex)	Meaning
00xx	Error Reset or No Error
10xx	Generic Error
20xx	Current
21xx	Current, device input side
22xx	Current inside the device
23xx	Current, device output side
30xx	Voltage
31xx	Mains Voltage
32xx	Voltage inside the device
33xx	Output Voltage
40xx	Temperature
41xx	Ambient Temperature
42xx	Device Temperature
50xx	Device Hardware
60xx	Device Software
61xx	Internal Software
62xx	User Software
63xx	Data Set
70xx	Additional Modules
80xx	Monitoring
81xx	Communication
8110	CAN Overrun (Objects lost)
8120	CAN in Error Passive Mode
8130	Life Guard Error or Heartbeat Error
8140	recovered from bus off
8150	Transmit COB-ID collision
82xx	Protocol Error
8210	PDO not processed due to length error
8220	PDO length exceeded
90xx	External Error
F0xx	Additional Functions
FFxx	Device specific

Tabla 2.6: Códigos de error para los mensajes de emergencia de CANopen.
Se muestran los diferentes códigos de error y su significado**.** Tabla obtenida de CiA **[8]**.

- **Error Register** (1 *byte*): entrada con índice 0x1001 del diccionario de objetos.

- **Manufacturer-specific Error Field** (5 *bytes*): este campo puede usarse para información adicional sobre el error. Los datos incluidos y su formato son definidos por el fabricante del dispositivo.

Los servicios de emergencia son opcionales pero en caso de emplearse deben al menos soportar los códigos de error 00xx y 10xx. Los demás errores son opcionales.

2.4.8 Mensajes Bootup

Estos mensajes indican al maestro que el dispositivo ha pasado del estado de inicialización al Pre-operacional. Este protocolo utiliza las mismas tramas que los protocolos de error. Véase Ilustración 2.15.

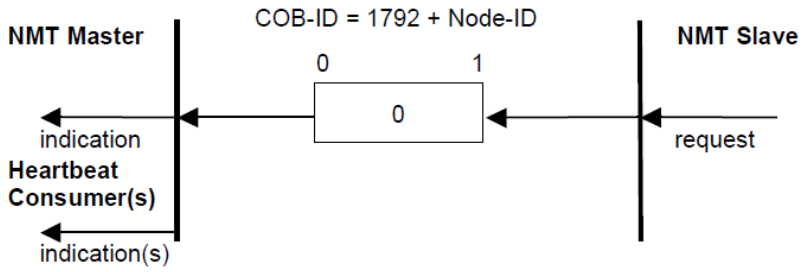


Ilustración 2.15: Protocolo Bootup.
Mediante el mensaje Bootup los dispositivos indican que han pasado el estado de inicialización y se encuentran en el Pre-operacional. Ilustración obtenida de CiA **[8]**.

2.4.9 Gestión de la red. Mensajes NMT (Network Management)

Se trabaja con un modelo de comunicaciones maestro-esclavo en el cual un dispositivo es el NMT maestro y el resto los NMT esclavos. Cada esclavo posee un único identificador denominado Node-ID que van desde el 1 al 127. El identificador 0 es propio del maestro. A través de los servicios NMT se pueden variar los estados en los que se encuentran los nodos de la red CANopen y con ello su funcionalidad.

Sólo el NMT maestro puede mandar mensajes de control NMT, pero todos los esclavos deben dan soporte a los servicios de control NMT. No hay respuesta para un mensaje NMT. Ver Ilustración 2.16.

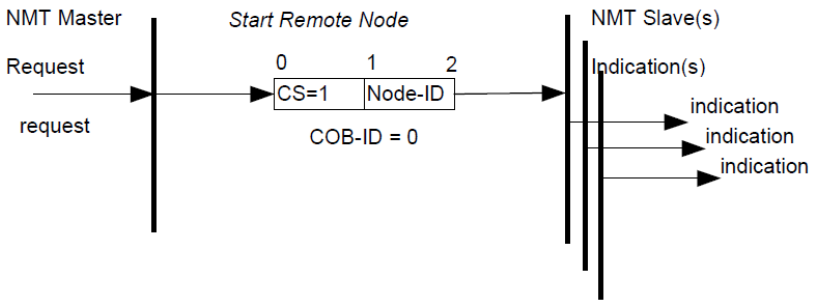


Ilustración 2.16 Protocolo NMT.
Protocolo utilizado por el maestro para el control de los esclavos. Ilustración obtenida de CiA **[8]**.

CS (command specifier)

- 1: Start
- 2: Stop
- 128: Pre-operational
- 129: Reinicio del Nodo
- 130: Reinicio de la Comunicación.

2.4.10 Máquina de estados NMT

Los diferentes estados que pueden alcanzar los nodos dentro de la red CANopen son *Initialisation*, *pre-operational*, *operational* y *stopped*. Al ser alimentados los dispositivos pasan del estado *Initialisation* al *pre-operational* automáticamente si no se producen fallos durante su fase de inicialización. Las demás transiciones entre estados solo pueden ser ordenadas por el maestro de la red. A continuación se explican los diferentes estados:

Initialisation: al encender el dispositivo se entra directamente a este estado. Después de realizar las labores de inicialización el nodo transmite el mensaje de *Boot-up* y pasa al estado Pre-Operacional.

Pre-operational: en este estado el dispositivo puede ser configurado mediante SDOs. Puede enviar y recibir SDOs, mensajes de emergencia, de sincronización, *time stamp* y mensajes NMT.

Operational: el dispositivo ya ha sido configurado y funciona normalmente. Todos los objetos de comunicación están activos así también puede enviar y recibir PDOs.

Stopped: todos los objetos de comunicación dejan de estar activos. No se pueden enviar ni recibir PDOs ni SDOs, sólo mensajes de NMT.

La Ilustración 2.17 muestra las posibles transiciones entre los diferentes estados y la Tabla 2.7 contiene las diferentes señales que son necesarias enviar para que se produzcan las transiciones:

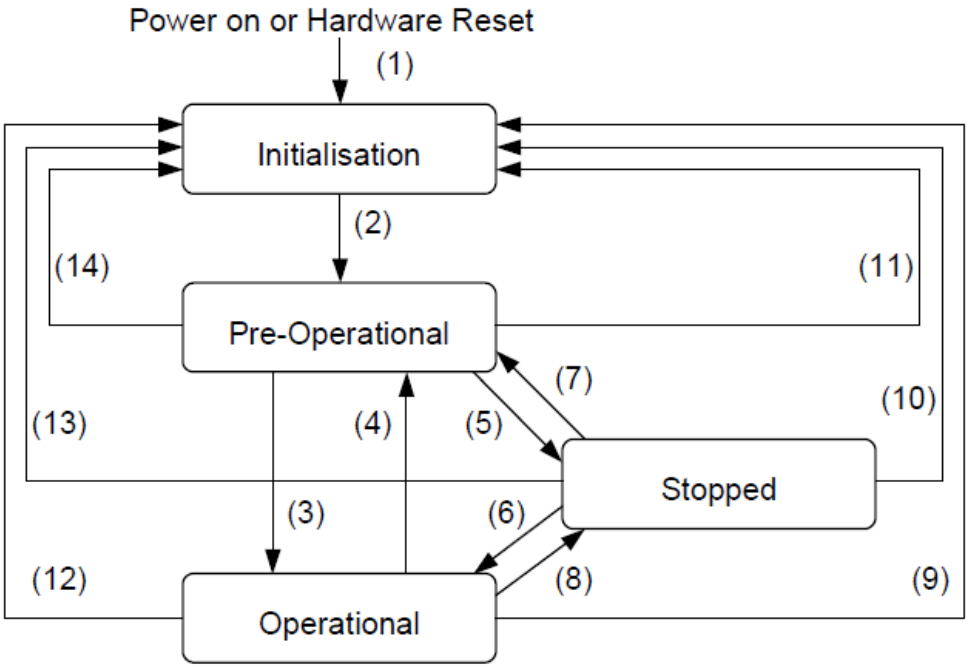


Ilustración 2.17. Máquina de estados CANOpen.
Muestra la máquina de estado de los nodos de una red CANopen junto con las posibles transiciones. El significado de la numeración se encuentra en Tabla 2.7. Ilustración obtenida de CiA **[8]**.

Nº Transición	Señal de transición
(1)	Alimentación del dispositivo o reinicio del hardware
(2)	Inicialización terminada y paso automáticamente al estado pre-operacional
(3),(6)	Señal “Start Remote Node”
(4),(7)	Señal “Enter_PRE-OPERATIONAL_State”
(5),(8)	Señal “Stop_Remote_Node”
(9),(10),(11)	Señal Reset_Node
(12),(13),(14)	Señal Reset_Communication

Tabla 2.7: Señales de transición de estados.
Contiene las señales necesarias para las transiciones entre estados. Tabla obtenida de CiA [8].

La Tabla 2.8 muestra las relaciones entre los estados de comunicación y las comunicaciones permitidas de los diferentes objetos que podrán llevarse a cabo o no dependiendo del estado en que se encuentre el dispositivo.

OBJETOS/ESTADOS	INITIALISING	PRE-OPERATIONAL	OPERATIONAL	STOPPED
PDO			X	
SDO		X	X	
SYNC		X	X	
Time Stamp Object		X	X	
Emergency Object		X	X	
Boot-Up Object	X			
Network Management Objects		X	X	X

Tabla 2.8: Comunicación de objetos permitida según el estado.
Muestra los diferentes tipos de mensajes que pueden ser enviados dependiendo del estado en que se encuentre el nodo. Tabla obtenida de CiA [8].

3. Componentes hardware y software utilizados en el proyecto

A continuación se presentan los elementos hardware utilizados para la instalación de una red CANopen. En concreto se describe el bus CAN, el dispositivo de control MCDC3006C que constituirá un nodo esclavo de la red y un adaptador PCAN-USB. Este adaptador permite la conexión de un ordenador, donde residen los nodos maestros, con el bus CAN.

Una vez explicado el hardware se presentara el software utilizado comenzando por la Interfaz de programación de aplicaciones o API (Application Programming Interface) denominada Canfestival que cubre la comunicación, mediante protocolo CANopen, desde los nodos maestros hasta los nodos esclavos de la red. Esta API facilita una serie de librerías que permiten el envío de las tramas CANopen e implementan todo el protocolo de comunicación CANopen. Posteriormente se describe el framework para el control y programación de robots denominado ROS (Robot Operating System) con el que se ha diseñado una arquitectura de control para el uso del controlador desarrollado por parte de otras aplicaciones.

3.1 Bus CAN y adaptador PCAN-USB

El bus CAN está constituido por dos cables trenzados que unen todos los nodos que forman el sistema. Los datos se trasmiten por diferencia de tensión entre los dos cables, de forma que un valor alto de tensión representa un 1 y un valor bajo de tensión representa un 0. El trenzado entre ambas líneas sirve para anular los campos magnéticos y las interferencias entre los propios cables. En los extremos de los cables se conectan resistencias, cuyos valores se obtienen de forma empírica y permiten adecuar el funcionamiento del sistema a diferentes longitudes de cables y número de nodos del sistema. Estos elementos finalizadores son utilizados para impedir fenómenos de reflexión que pueden perturbar el mensaje.

En un cable los valores de tensión oscilan entre 0V y 2.25V, por lo que se denomina cable L (Low) y en el otro, el cable H (High) lo hacen entre 2.75V y 5V. En caso de que se interrumpa la línea H o que se derive a masa, el sistema trabajará con la señal de Low con respecto a masa. En el caso de que se interrumpa la línea L, ocurrirá lo contrario. Esta situación permite que el sistema siga trabajando con uno de los cables cortados o comunicados a masa, incluso con ambos comunicados también sería posible el funcionamiento, quedando fuera de servicio solamente cuando ambos cables se cortan.

Existen convertidores como el de la Ilustración 3.1 que permiten conectar un ordenador mediante su puerto USB al bus de comunicación CAN mediante un conector D-Sub de 9 pines. En este proyecto se ha utilizado dicho convertidor para conectar la unidad de control del robot Mopi al bus de comunicación CAN donde están también conectados los dispositivos controladores de los motores permitiendo el envío de las órdenes.

Ilustración 3.1. Adaptador PCAN-USB.
Este adaptador ha sido utilizado para conectar la unidad central de control del robot con el bus CAN. Ilustración obtenida de Peak System **[10]**.

3.2 Dispositivo de control Faulhaber MCDC3006C

La base del robot Mopi utiliza dos motores de corriente continua o DC (Direct Current) por lo que son necesarios dos dispositivos controladores, uno por cada motor, que permitan su control mediante el protocolo CANopen. En concreto, el dispositivo escogido es el MCDC3006C del fabricante Faulhaber [11]. Ver Ilustración 3.2. Este dispositivo permite el control de motores DC, de hasta 30v y 6A, mediante el protocolo de comunicación RS232 o CANopen. Dispone de entradas para señales de encoder permitiendo el control de velocidad, posición y par. Se pueden controlar varias unidades a la vez mediante sistema en red, ya sea por RS232 o mediante comunicación vía CAN, cuya conexión se realiza mediante un conector D-Sub de 9 pines. Además es posible programar perfiles de funcionamiento, visualizar en tiempo real graficas de consumo y velocidad mediante las herramientas software suministradas por el fabricante.

Ilustración 3.2: Driver MCDC3006C.
Se trata de un dispositivo de control de motores mediante el protocolo CANopen. Ilustración obtenida de Faulhaber **[11]**.

3.2.1 Firmware CANopen del dispositivo de control MCDC3006C

Como se ha comentado, este dispositivo permite su integración en una red CANopen con velocidades de transferencia de hasta 1 Mbit/s. Cumple las especificaciones de comunicación CANopen establecidas en el documento DS301 V4.02 [8] así como las especificaciones de los dispositivos de control de motores descritas en el documento DSP402 V2.0 [9]. Dispone de tres modos de funcionamiento, por posición, velocidad o homing mode que es utilizado en la fase de calibración. Véase Ilustración 3.3.

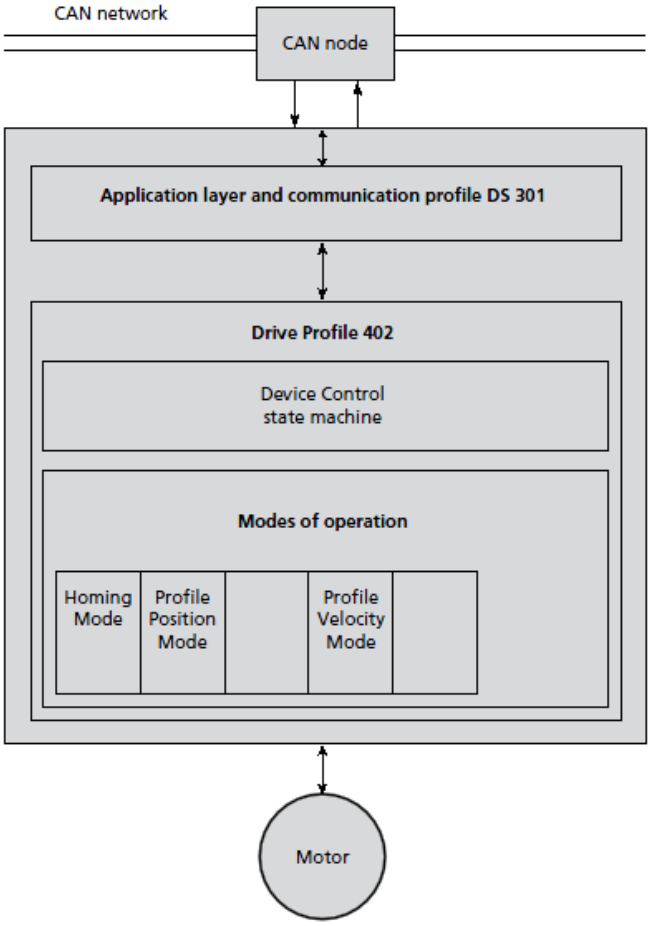


Ilustración 3.3: Firmware MCDC3006C.
El firmware CANopen cumple con las especificaciones de la CiA **[7]** en los documentos DS402 y DS301. Ilustración obtenida de **[12]**.

Este driver dispone de los siguientes tipos de objetos:

- Objetos SDO (Service Data Objects): Definen un mensaje SDO de transmisión y otro de recepción.

- Objetos PDO (Process Data Objects): Definen tres mensajes PDO de transmisión y tres de recepción. Solo el mensaje PDO1 está especificado en el documento de la CiA DS402 [9]. Parámetros como la corriente consumida, temperatura, máximo voltaje, etc, debían estar definidos mediante objetos en el diccionario de objetos y poder ser leídos directamente pero el fabricante no cumple totalmente con el

protocolo CANopen. Por ello implementa los mensajes de transmisión y recepción PDO2 y PDO3, al margen de dichas especificaciones, para la lectura de dichos parámetros.

- Objeto NMT con Node Guarding: Define el mensaje Node Guarding.
- Objeto Emergency: Define los mensajes de emergencia.
- Objeto SYNC: Define los mensajes de sincronización utilizados para la comunicación síncrona mediante mensajes PDO.

A continuación se detallan los mensajes definidos por los objetos PDO:

3.2.1.1 **Mensajes PDO**

Como se ha comentado anteriormente el dispositivo de control de Faulhaber dispone de tres mensajes PDO de transmisión (TxPDO) y tres mensajes de recepción (RxPDO).

Mensajes TxPDO1 y RxPDO1

El mensaje TxPDO1 permite transmitir el objeto *Statusword* (0x6041) que es utilizado para visualizar el estado del motor, por ejemplo si está habilitado, inhabilitado, si ha alcanzado la velocidad que se le ha indicado, etc. Con el mensaje RxPDO1 se recibe el objeto *Controlword* (0x6040) utilizado para cambiar el estado del motor y poder por ejemplo habilitarlo o deshabilitarlo. Véase Ilustración 3.4.

TxPDO1: *Statusword*

11 bit identifier	2 bytes user data	
0x180 (384D) + node ID	LB	HB

RxPDO1: *Controlword*

11 bit identifier	2 bytes user data	
0x200 (512D) + Node-ID	LB	HB

Ilustración 3.4 Objetos RxPDO1 y TxPDO1 del driver mcdc3006c.
Se muestra la estructura de los mensajes TxPDO1 y RxPDO1. Ilustración obtenida de Faulhaber **[12]**.

Mensajes TxPDO2 y RxPDO2

Los mensajes TxPDO2 y RxPDO2 son utilizados para el envío de comandos propios del fabricante para realizar las diferentes funciones. El objeto que transportan los mensajes TxPDO2 se denomina *FaulhaberData* (0x2303) y el mensaje RxPDO transporta el objeto *FalulhaberCommand* (0x2304). Se pueden observar que ambos objetos se encuentran en las entradas del diccionario denominadas Manufacturer Specific ya que son objetos propios del fabricante. La estructura de los mensajes PDO2 puede observarse en la Ilustración 3.5.

TxPDO2: FAULHABER data

11 bit identifier	6 bytes user data					
0x280 (640D) + Node-ID	Cmd	LLB	LHB	HLB	HHB	Error

RxPDO2: FAULHABER command

11 bit identifier	5 bytes user data				
0x300 (768D) + Node-ID	Cmd	LLB	LHB	HLB	HHB

Ilustración 3.5 Objetos RxPDO2 y TxPDO2 del driver mcdc3006c.
Se muestra la estructura de los mensajes TxPDO2 y RxPDO2. Ilustración obtenida de Faulhaber **[12]**.

Mensajes TxPDO3 y RxPDO3

Mediante los mensajes TxPDO3 y RxPDO3 pueden conocerse algunos datos como los valores de temperatura del driver, corriente utilizada, temperatura, etc. Puesto que estos datos no han sido establecidos como indican las especificaciones DS402 de la CiA, la utilización de este mensaje es la única forma de obtenerlo. Mediante el mensaje RxPDO3 se recibe el objeto *TraceConfiguration* (0x2305) y con el mensaje TxPDO3 se envía el objeto *TraceData* (0x2306). El fabricante establece en sus especificaciones que comandos deben ser enviados mediante el objeto *TraceConfiguration* para obtener el valor de respuesta en el objeto *Trace Data*. Ambos objetos se encuentran en el diccionario de objetos en el apartado *Manufacturer Specific*. Ver Ilustración 3.6.

TxPDO3: Trace data

11 bit identifier	3 to 8 byte user data							
0x380 (896D) + Node-ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7

RxPDO3: Trace configuration

11 bit identifier	5 bytes user data				
0x400 (1024D) + node ID	Mode 1	Mode 2	TC	Packets	Period

Ilustración 3.6 Objetos RxPDO3 y TxPDO3 del driver mcdc3006c.
Se muestra la estructura de los mensajes TxPDO3 y RxPDO3. Ilustración obtenida de Faulhaber **[12]**.

3.2.1.2 Máquina de estados del driver MCDC3006C

Como se establece en el perfil DS402 de la CiA [9], los dispositivos de control disponen de una máquina de estados interna como se puede observar en la Ilustración 3.7. Dependiendo en el estado en que se encuentren, podrán realizar determinadas funciones. Además puesto que este dispositivo constituye un nodo en la red CANopen sus comunicaciones están determinadas por la máquina de estados propia de la red CANopen (ver apartado 2.4.10).

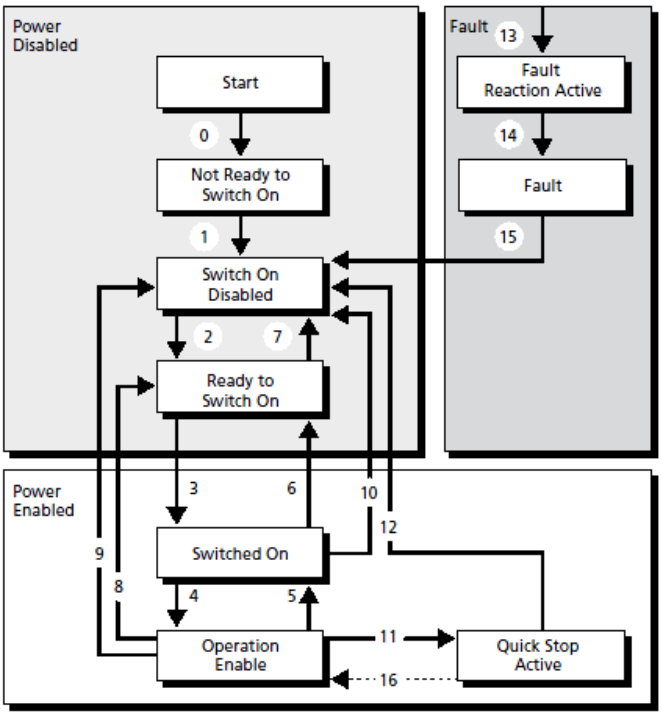


Ilustración 3.7. Máquina de estados driver MCDC3006C.
Muestra los diferentes estados que definen el funcionamiento de los dispositivos de control de motores. Ilustración obtenida de Faulhaber **[12]**.

Al alimentarse el dispositivo se llega automáticamente hasta el estado **Switch On Disable.** No se puede realizar ningún cambio de estado hasta que el dispositivo este en estado **Operacional** dentro de la red CANopen. Una vez que se encuentra en estado operacional se puede modificar el estado interno mediante el envío de diversas señales a través del objeto *Controlword*. La Ilustración 3.7 junto con la Tabla 3.1 permite observar las diferentes señales necesarias para producir las transiciones entre estados. Para conocer los valores de dichas señales debe consultarse el manual del fabricante Faulhaber [12].

Señal de transición	Transición
Shutdown	**2,6,8**
Switch on	**3**
Disable Voltage	**7,9,10,12**
Quick Stop	**7,10,11**
Disable Opertaion	**5**
Enable Operation	**4,16**
Fault Reset	**15**

Tabla 3.1.Comandos de transición entre estados del driver MCDC3006C.
Señales necesarias para la transición entre estados. Ilustración obtenida de Faulhaber **[12]**.

En el estado **Switched On** el motor está alimentado pero no ejecuta las órdenes de posición y velocidad ya que se encuentra inhabilitado. El funcionamiento normal del driver se lleva a cabo en el estado **Operation Enable.**

3.3 Canfestival

Canfestival es una plataforma de programación C (ANSI-C) que da soporte de comunicación mediante el protocolo CANopen así como la creación de una aplicación que constituya un nodo esclavo o un maestro de la red. Permite la creación simulada de una red CANopen sin necesidad del uso de hardware. Se trata de una plataforma de código abierto bajo la licencias LGPL (Lesser General Public License) y GPL (General Public License). Fue iniciada por Edouard Tisserant en 2001 y ha seguido creciendo gracias a la colaboración de Francis DUPIN entre otros. Actualmente es un proyecto que sigue evolucionando gracias al respaldado del grupo Lolitech [13].

- **Funcionamiento:**

La aplicación maestro/esclavo debe ser implementada por el usuario. En caso de ser un maestro, debe abrir el puerto CAN e iniciar los temporizadores que permite controlar las comunicaciones. Posteriormente se inicializaran los nodos esclavos. Opcionalmente tanto para el maestro como para los esclavos se pueden establecer su funcionamiento para cada uno de sus diferentes estados (pre-operacional, operacional, etc.).

La aplicación que represente un nodo maestro/esclavo debe contar además con un diccionario de objetos. Este será diseñado con una herramienta proporcionada por Canfestival denominada ***objDictedit*** [3] que posteriormente será explicada en el anexo B. Esta aplicación utilizará la librería Canfestival para la gestión de los nodos (acceso al diccionario de objetos y máquina de estados) y el envío de los diferentes tipos de mensajes ya que en ella se encuentran definidos los diferentes protocolos de comunicación (mensajes SYNC, SDO, PDO, NMT, LSS). Lo comentado anteriormente puede observarse en la Ilustración 3.8.

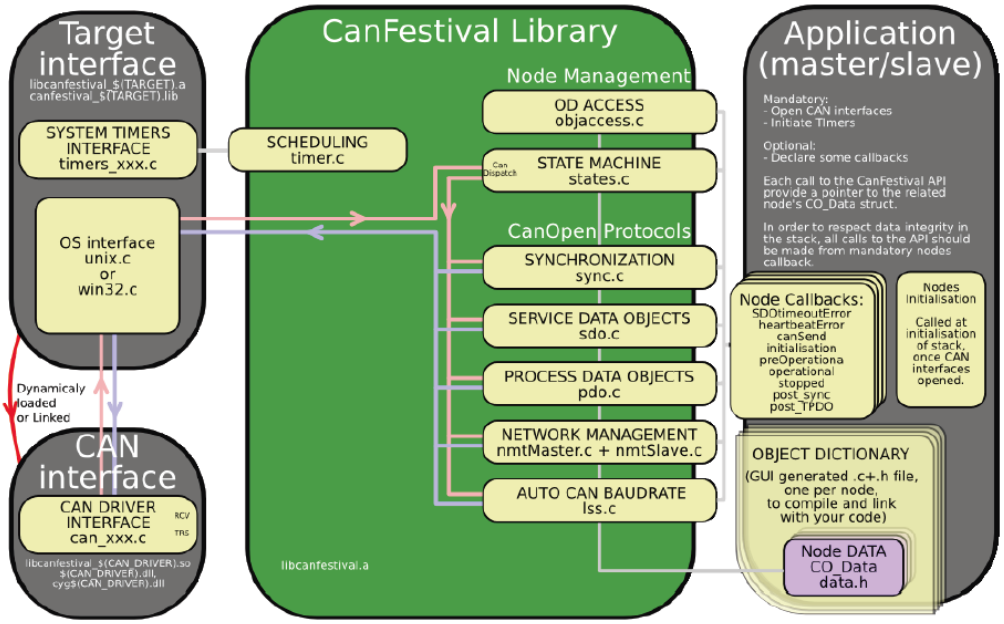


Ilustración 3.8: Diagrama Canfestival.
Canfestival suministra todas las herramientas para la comunicación CANopen. Ilustración obtenida de Canfestival **[3]**.

La Ilustración 3.9 muestra cómo se debe crear una aplicación que permita diseñar un maestro para la red CANopen o simule un nodo esclavo.

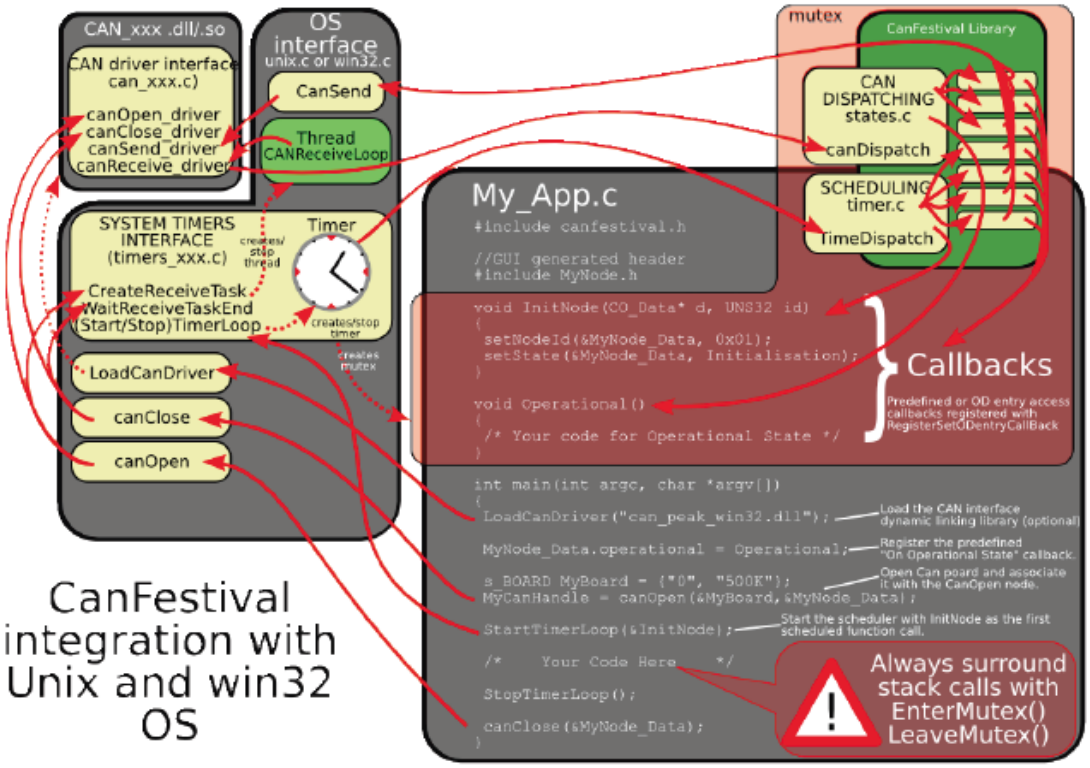


Ilustración 3.9: Canfestival. Estructura de una aplicación.
Muestra los bloques que deben contener una aplicación maestra o esclavo. Ilustración obtenida de Canfestival **[3]**.

3.4 ROS (Robot Operating System)

ROS (Robot Operating System) [14] es un framework utilizado en robots. Suministra servicios propios de un sistema operativo como la abstracción hardware, el control de dispositivo a bajo nivel, implementación de funcionalidades comunes como envío de mensajes entre procesos y la administración de paquetes. También se suministran herramientas y librerías para el desarrollo de aplicaciones software en robots. En este proyecto se utiliza ROS para integrar el controlador CANopen dentro de una red ROS que le permita la comunicación con las diferentes aplicaciones. Esta red ROS se explicará en el apartado 4.

Unidades básicas de un sistema ROS:

ROS permite diferentes tipos de comunicación como la comunicación síncrona RPC mediante **Services**, comunicación asíncrona a través de los **Topics** y almacenamiento de datos mediante **Parameter Server**. ROS está distribuido bajo la licencia de código abierto BSD *(Berkeley Software Distribution).* Las unidades básicas del sistema ROS son: Nodes, Master, Messages, Topics, Services.

- **Nodes:** Los nodos son la estructura más pequeña de una red ROS. Se comunican entre sí utilizando los topics de transmisión, servicios RPC y Server Parameter. Un sistema de control de un robot utiliza un gran número de nodos para el desarrollo de las diferentes habilidades, lo que demuestra la gran modularidad de ROS. Imagínese por ejemplo un robot que debe moverse por un entorno con obstáculos. Su sistema de control estaría compuesto por varios nodos de los cuales uno podría encargarse de la recopilación de datos de los sensores que permiten al robot detectar obstáculos. Estos datos recogidos serían enviados a un nodo de control donde se procesarían y se ejecutarían las órdenes oportunas. Después de procesar los datos podría determinarse que el robot tiene un obstáculo cercano, por tanto, se enviaría al nodo encargado del control de los motores, que envíe la orden para que estos se muevan de forma que permitan esquivar dicho obstáculo. Podría existir un cuarto nodo que se encargue de mostrar por pantalla la información importante, como el valor de los sensores.

- **Master:** El ROS Master es el servidor central de la arquitectura, su principal tarea es permitir que los nodos se localicen entre ellos para que puedan comunicarse mediante los Topics y Services. También suministra los Parameter Server.

- **Messages:** Los nodos se comunican entre ellos mediante la publicación de mensajes que transmiten mediante Topics o Services. Un mensaje es una estructura simple de diferentes tipos de datos. Los tipos básicos son enteros, decimales, booleanos, etc. También soportan vectores y estructuras de datos.

- **Topics:** Los topics son las vías de intercambio de mensajes entre nodos mediante un sistema de publicación y suscripción. Los nodos se suscriben a un determinado topic donde se publican los datos que desean conocer o publican datos en otro para para ser utilizados por otros nodos. Pueden existir varios publicadores y suscriptores de un mismo topic. Los nodos desconocen con que otros nodos se están comunicando. La comunicación mediante topics es unidireccional.

  Volviendo al ejemplo anterior, el nodo encargado del control de los motores, estaría suscrito a un topic denominado *controlMotors*, en el que el nodo de procesamiento de datos publicase órdenes de movimiento de motores. Este nodo de procesamiento a su vez estaría suscrito al topic *dataSensors* mediante el cual el nodo de adquisición de datos publicaría los datos recogidos.

- **Services:** Los sistemas de comunicación petición/respuesta se realizan mediante servicios, los cuales están definidos por un par de mensajes: uno para la petición y otro para la respuesta. Utiliza el modelo de comunicación cliente/servidor.

Unidades de almacenamiento:

El código implementado mediante el framework ROS se agrupa en Packages y Stacks para facilitar su reutilización. A continuación se describen ambas estructuras de organización:

- **Packages:** El software de ROS está organizado en packages o paquetes. Cada paquete posee una función propia permitiendo una arquitectura modular que facilita el uso del software implementado. Un paquete puede contener los diferentes nodos ROS, librerías, archivos de configuración, software de terceras partes, etc. El ejemplo anterior podría estar compuesto por diferentes paquetes. Imagínese que el código que ejecuta cada nodo estuviese agrupado en un paquete diferente. Se dispondría de un paquete donde se han implementado la unidad de lectura de los sensores, otro paquete almacena el código de control de los motores, otro la unidad de control y procesamiento de datos y un último paquete contiene todo el código necesario para la monitorización de la información. Esta organización facilita la reutilización del código. En caso de que se quiera diseñar otro robot móvil con diferente funcionalidad podrían reutilizarse los paquetes de control de motor y recopilación de datos y diseñarse un paquete diferente de control y procesamiento de datos.

- **Stacks:** Los paquetes ROS están organizados en Stacks. Mientras que el propósito de los packages es facilitar la reutilización del código implementado el de los Stacks es facilitar el proceso distribución. Los Stacks engloban packages que juntos suministran una funcionalidad, por ejemplo puede existir el stack *Automoción* que suministra todos los paquetes que permiten que el robot se mueva.

4. Desarrollo del proyecto

El objetivo principal del proyecto es el desarrollo de un controlador CANopen, implementando sus diferentes funcionalidades, para permitir el control de los motores del robot Mopi. Además, para permitir el uso por parte de las diferentes aplicaciones del robot, se ha diseñado la arquitectura de software utilizando el framework de ROS. En este apartado se explicará detalladamente el funcionamiento de dicho controlador CANopen y su arquitectura software diseñada. Al final del capítulo se describen los pasos de configuración necesarios para la adición de un nuevo controlador CANopen.

Para la verificación del correcto funcionamiento del controlador se ha realizado una prueba inicial, previa al montaje en el robot, para la cual se ha desarrollo un módulo de control y otro de verificación de datos. Estos módulos permiten simular el comportamiento que tiene el controlador durante la comunicación con terceras aplicaciones. Una vez comprobado el correcto funcionamiento del controlador, se procede a la instalación hardware de la red CANopen en el robot Mopi y la integración software del controlador en su CPU (Central Processing Unit).

4.1 Arquitectura software de comunicación

Las órdenes enviadas, desde una aplicación hasta su ejecución por los motores, pasan por dos redes: La red ROS y la red CANopen. El software del controlador CANopen se ha diseñado de manera que constituya un nodo de ROS, dentro de la red ROS, y un nodo maestro, dentro de la red CANopen. Ver Ilustración 4.1.

La red ROS permite la comunicación de los diferentes controladores CANopen con otros nodos ros que serán las diferentes aplicaciones. Esto permite que las aplicaciones envíen las órdenes a los controladores CANopen y la lectura de la información de los motores recogida por éstos.

En la red CANopen se establecen las comunicaciones de los controladores CANopen con los dispositivos físicos MCDC3006C. Cada controlador CANopen es un nodo maestro y tiene asignado un único nodo esclavo representado por el dispositivo MCDC3006C. Los dispositivos de control MCDC3006C son los encargados de ejecutar las órdenes procedentes de los nodos maestros. Los nodos maestros se encuentran en el software implementado y los nodos esclavos son los dispositivos hardware de control de motores.

La operatividad de cada motor se establece mediante un controlador CANopen y un dispositivo físico MCDC3006C. Por tanto, el número de nodos maestros y de nodos esclavos es igual al número de motores.

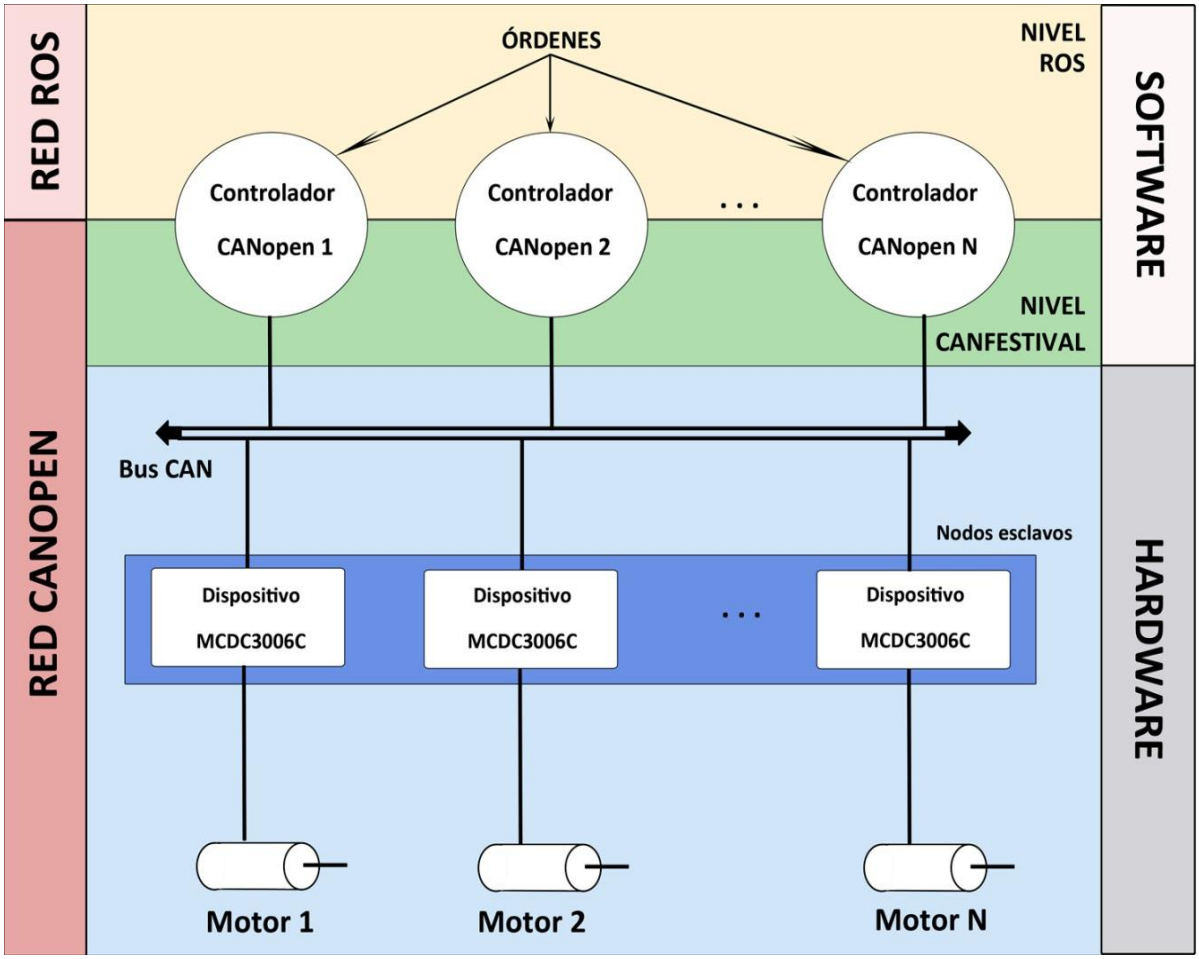


Ilustración 4.1. Estructura de comunicación.
Las órdenes son enviadas hasta los controladores mediante la red ROS para posteriormente ser transformadas en tramas CANopen y enviadas hasta los dispositivos de control mediante la red CANopen. Por diseño a cada motor tiene asignado un controlador CANopen.

Como se explicó en el apartado 2.4, el protocolo CAN es la base del protocolo CANopen, estableciendo las capas física y de enlace del modelo OSI. El protocolo CANopen define la capa de aplicación del modelo OSI permitiendo la abstracción de las tramas CAN, por lo que en esta memoria solo se explicarán las tramas a nivel CANopen.

4.2 Controlador CANopen

El controlador CANopen, que a partir de ahora se llamará ***ROSCanfestivalNode***, está constituido por los niveles ROS y Canfestival como se observa en la Ilustración 4.2. *ROSCanfestivalNode* es un nodo en la red ROS y un nodo maestro dentro de la red CANopen.

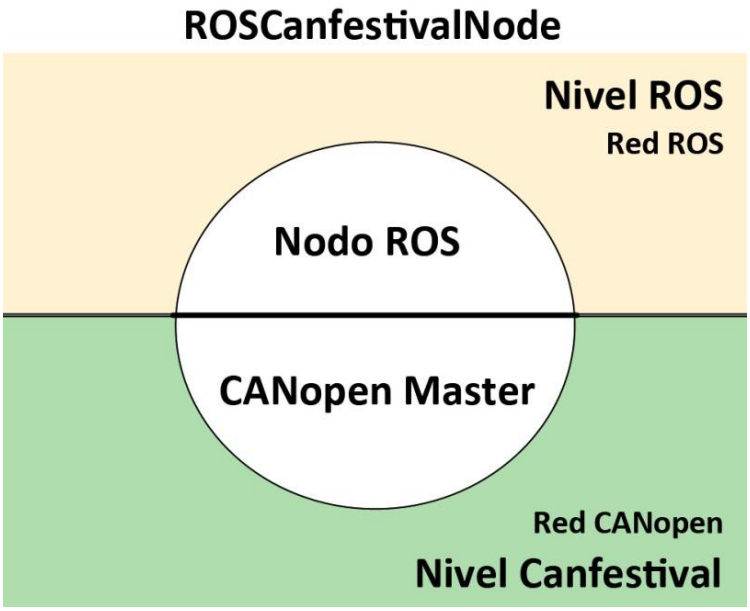


Ilustración 4.2. Estructura software de ROSCanfestivalNode.
El controlador CANopen está estructurado en el nivel ROS y el nivel Canfestival. Constituye un nodo de la red ROS y un nodo maestro de la red CANopen.

El funcionamiento del controlador CANopen es el siguiente: las órdenes de control son recibidas en el nivel ROS y convertidas en tramas CANopen dentro del nivel Canfestival para ser enviadas a los dispositivos de control de motores MCDC3006C a través del bus CAN. Mientras no se reciban órdenes de funcionamiento el controlador *ROSCanfestivalNode* realiza una petición continua de datos de sensor, configuración y estado del motor al dispositivo de control de motores MCDC3006C. Cuando se termina el bucle de petición se comprueba si existen órdenes, de ser así, estas son enviadas y comienza otra vez el bucle de petición. La Ilustración 4.3 muestra el diagrama de casos de uso de *ROSCanfestivalNode*.

En su nivel ROS, *ROSCanfestivalNode* es un nodo de ROS lo que le permite comunicarse con los demás nodos de la red en los que, por ejemplo, se ejecuten las diferentes habilidades de un robot. En este nivel, *ROSCanfestivalNode* realiza la suscripción a los topics, a través de los cuales llegarán las órdenes, que deben cumplir los motores, desde los diferentes nodos de la red. Del mismo modo *ROSCanfestivalNode* publica los datos de funcionamiento recogidos del motor en los topics de salida para ser utilizados por los demás nodos. El nivel ROS se explica con mayor profundidad en el apartado 4.3.

En el nivel Canfestival, para poder enviar datos por el bus CAN, se comienza con la inicialización de las comunicaciones para que *ROSCanfestivalNode* constituya un nodo

maestro CANopen de la red CANopen. La inicialización del maestro se lleva a cabo gracias a la API de Canfestival. Además, en este nivel se convierten las órdenes recibidas a través de los topics en tramas CANopen para ser enviadas a los nodos esclavos MCDC3006C. Del mismo modo, las tramas CANopen procedentes de los nodos esclavos son procesadas para obtener los diferentes datos de funcionamiento de los motores. La comunicación y configuración de la red CANopen se explica en detalle en el apartado 4.4.

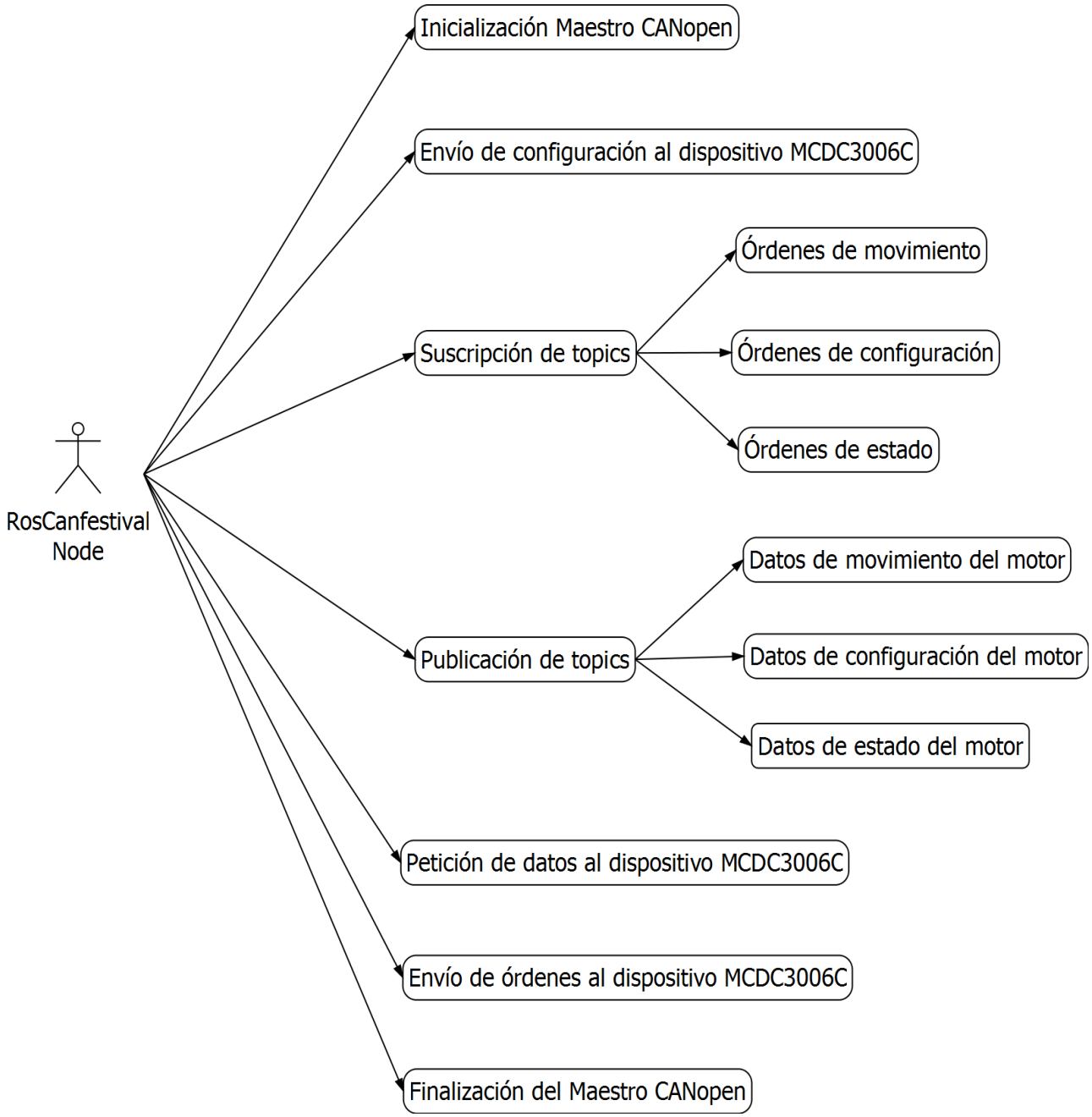
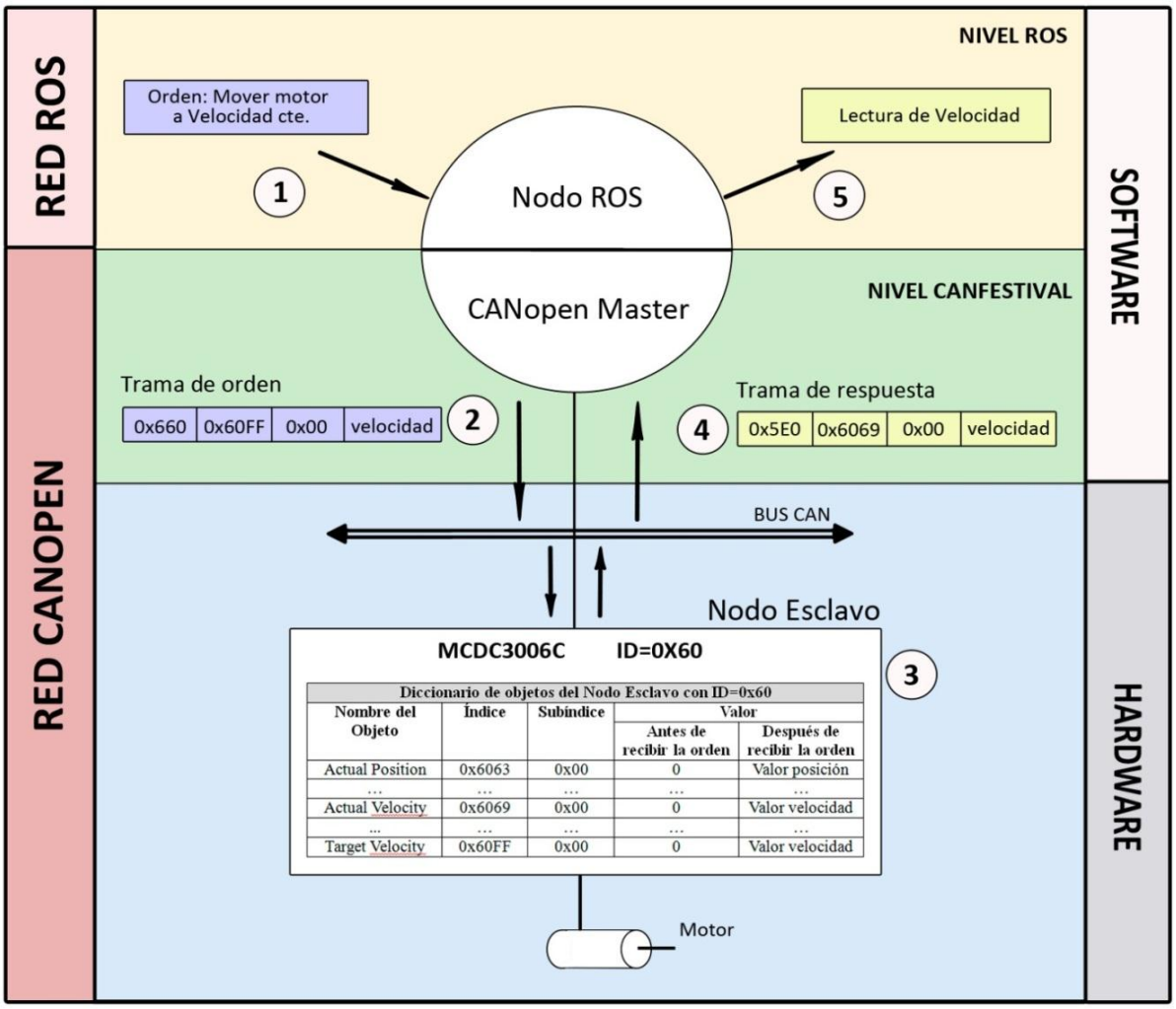


Ilustración 4.3. Diagrama de casos de Uso de ROSCanfestivalNode.
Las diferentes tareas realizas por *ROSCanfestivalNode* se encuentra reflejadas en el diagrama de casos de uso.

El funcionamiento del nodo *ROSCanfestivalNode* comienza con la inicialización de las comunicaciones y del maestro CANopen estableciéndole en estado Operacional lo que le permite la comunicaciones mediante mensajes SDO y PDO. El nodo esclavo MCDC3006C alcanza el estado pre-operacional (*pre-operationa)* al ser alimentados pudiendo únicamente comunicarse mediante mensajes SDO. Por tanto, el nodo maestro, *ROSCanfestivalNode,* debe establecer el estado operacional (*operational)* al nodo esclavo para permitir su total comunicación. Establecidas las comunicaciones y el estado operacional del esclavo es posible el envío de todo tipo de tramas CANopen a través del bus. *ROSCanfestivalNode* comenzará enviando los datos de configuración del motor al nodo esclavo MCDC3006C. Los valores de configuración son facilitados mediante un archivo de configuración XML. Un ejemplo de este archivo de configuración puede observarse en el anexo D. A partir de este punto *ROSCanfestivalNode* entra en un bucle de envío de tramas de petición de datos al nodo esclavo que serán contestadas de forma asíncrona. Una vez finalizada la primera iteración del bucle de petición de datos, *ROSCanfestivalNode* comprueba si existen órdenes en los topics a los que está suscrito. En caso de existir órdenes pendientes pasarán al nivel Canfestival donde serán convertidas en tramas CANopen y enviadas a través del bus. El funcionamiento del bucle de petición se realiza a una frecuencia de 10Hz por lo que la comprobación de cambios en los topics de entrada es prácticamente instantánea, no suponiendo retrasos en la ejecución de las órdenes.

El bucle se mantiene hasta que se destruye el nodo ROS, momento en el que se cierran las comunicaciones, liberando y reseteando los nodos esclavos así como la finalización del maestro CANopen.

El ejemplo de la Ilustración 4.4 describe el envío de una orden procedente de una habilidad del robot desarrollada en el framework ROS. Puede pensarse, por ejemplo, en una habilidad que permita al robot acercarse a la persona que tiene delante cuando esta se aleje. Cuando el robot detecte que la distancia con la persona está aumentando se enviará una orden desde la habilidad hasta el controlador CANopen para mover los motores por velocidad. El controlador CANopen procesa la orden de movimiento por velocidad y es enviada al dispositivo de control MCDC3006C quien deberá ejecutarla para que el motor comience a moverse. *ROSCanfestivalNode,* el controlador CANopen, constituye un nodo ROS, dentro la red ROS, y un nodo maestro de la red CANopen. En la red CANopen, el nodo maestro tiene asignado un nodo esclavo, representado por el dispositivo MCDC3006C, cuyo número de identificador es ID=0x60. Antes del envío de la orden el motor se encuentra detenido en su posición de inicio.



Diccionario de objetos del Nodo Esclavo con ID=0x60

Nombre del Objeto	Índice	Subíndice	Valor	
			Antes de recibir la orden	Después de recibir la orden
Actual Position	0x6063	0x00	0	Valor posición
...	...	...	...	...
Actual Velocity	0x6069	0x00	0	Valor velocidad
...	...	...	...	...
Target Velocity	0x60FF	0x00	0	Valor velocidad

Ilustración 4.4. Esquema de comunicación del controlador desarrollado.
El controlador recibe órdenes de alto nivel, a través de Topics, que traduce en tramas CANopen para ser enviadas a través del bus hasta los dispositivos de control MCDC3006C donde serán ejecutadas. También se encarga de solicitar datos del sensor, configuración y estado del motor.

A continuación se describen cada uno de los pasos numerados en la Ilustración 4.4.

1. Envío de la orden al controlador CANopen. Una aplicación o habilidad establece que el motor debe moverse a una velocidad constante. Para ello, la aplicación o habilidad, que constituye un nodo ROS, envía la orden de alto nivel a través de un topic al nodo *ROSCanfestivalNode* quien se encargará de procesar la orden.

2. Funcionamiento del controlador. *ROSCanfestivalNode* ejecuta un bucle de petición de datos de funcionamiento del motor a los dispositivos MCDC3006C. Una vez recibidos estos datos, son procesados y preparados para ser utilizados por las aplicaciones pertinentes. Puesto que el motor se encuentra detenido en su posición inicial se reciben los valores de velocidad cero y posición cero.

   Una vez terminado el bucle de peticiones se comprueba si hay órdenes que deban ser procesadas. Si se ha recibido alguna orden se convierte a una trama CANopen, que a su vez la API de Canfestival convierte en una trama CAN, y es enviada a

través del Bus CAN. Puesto que la API de Canfestival permite la utilización del protocolo CANopen, consiguiendo la abstracción del protocolo CAN, únicamente se muestra en la Ilustración 4.4 los elementos que constituyen las tramas CANopen.

En primer lugar la trama CANopen debe contener el identificador de mensaje COBID (Communication Object Identifier). Como se explicó en el apartado 2.4.4.1 el COBID depende del tipo de mensaje y del identificador del nodo al que va enviado. En este caso puesto que se envía una orden del nodo maestro al esclavo el COBID del mensaje de transmisión SDO (TxPDO) del maestro debe coincidir con el COBID del mensaje de recepción SDO (RxPDO) del esclavo que ha sido establecido por defecto (ver Tabla 2.4). El dispositivo MCDC3006C tiene un identificador *ID = 0x60* por lo que el valor del identificador de este mensaje será *COBID = 0x600 + ID = 0x660*. Además, la trama CANopen debe contener el índice y subíndice de la entrada del diccionario de objetos del dispositivo MCDC3006C en la que se desea escribir o leer. En este ejemplo se desea cambiar la velocidad del motor por lo que deberá escribirse en el objeto denominado *Target Velocity,* del diccionario del nodo esclavo, encargado de establecer la velocidad del motor. La entrada del diccionario que corresponde al objeto *Target Velocity* tiene el índice 0x60FF y subíndice 0x00. Por último la trama CANopen debe contener el valor de dicho objeto, en este caso la velocidad del motor. Cuando la orden se ha enviado comienza de nuevo el bucle de petición.

3. En la Ilustración 4.4 se observan los objetos Actual Position, Actual Velocity y Target Velocity, del diccionario del nodo esclavo MCDC3006C, que almacenan los valores de la posición, velocidad y velocidad a alcanzar del motor respectivamente. Inicialmente el motor se encontraba detenido en la posición inicial. Una vez llegue la orden se sobrescribirá el objeto *Target Velocity* y el motor comenzará a moverse a la velocidad indicada. En movimiento los valores de los objetos *Actual Position* y *Actual Velocity* se sobrescribirán con los valores obtenidos del encoder del motor.

4. Una vez que *ROSCanfestivalNode* ha convertido la orden en una trama CANopen y ha sido enviada a través de la red CANopen comienza de nuevo el bucle de petición de datos de funcionamiento del motor. Para la recopilación de datos *ROSCanfestivalNode* envía una serie peticiones, mediante mensajes del tipo SDO, entre las que se solicita el valor de velocidad del motor. Estas peticiones son enviadas al nodo esclavo MCDC3006C. El nodo esclavo contestará a dichas peticiones con la transmisión de mensajes tipo SDO. La petición del valor de la velocidad se contesta mediante un mensaje SDO en el que se incluye el valor de la velocidad almacenado en el objeto *Actual Velocity* del diccionario del nodo esclavo MCDC3006C. El identificador por defecto para mensajes de transmisión o TxSDO es *COBID = 0x580 +ID*, por tanto, el nodo esclavo con *ID=0x60* enviará un mensaje con *COBID = 0x580 + 0x60 = 0x5E0*. Esta trama de respuesta contiene el índice y subíndice del objeto que transporta, para el objeto *Actual Velocity* su índice es 0x6069 y su subíndice 0x00.

5. Procesamiento de la trama CANopen y preparación de su salida. Las tramas recibidas con los datos del sensor, configuración y estado del motor son procesadas y preparada su salida para su utilización por parte de otra aplicación.

4.3 Nivel ROS.

En el nivel ROS se crea una red ROS así como los diferentes nodos que representan los controladores CANopen implementados. En esta red el controlador *ROSCanfestivalNode* representa un nodo que se comunica con otros nodos de la red, en los que se ejecutaran las diferentes habilidades o aplicaciones.

En este apartado se describe en detalle la red ROS creada. Se comienza explicando la red ROS básica diseñada para el control de un solo motor. Una vez entendido su funcionamiento será fácil la incorporación de nuevos motores y la ampliación de la red ROS a partir de dicha red básica. En concreto, se detallará la red creada para el control de la base del robot Mopi, que consta de dos motores, mediante un mando inalámbrico o joypad.

4.3.1 Red ROS para un motor.

En este apartado se expone como el controlador *ROSCanfestivalNode* se comunica con los nodos de la red permitiendo la recepción de las órdenes y publicación de los datos de funcionamiento de los motores. La comunicación entre nodos ROS se ha establecido mediante topics, los cuales permiten una comunicación unidireccional. Los topics han sido descritos en el apartado 3.4. Por tanto, se han creado tres topics de entrada a través de los cuales serán enviadas las diferentes órdenes de control al controlador. Para la recepción de las órdenes el nodo *ROSCanfestivalNode* está suscrito a estos topics. Además se crean otros tres topics de salida donde *ROSCanfestivalNode* publica los datos procedentes del motor donde las demás aplicaciones se suscribirán para poder utilizarlos. Esta red ROS puede observarse en Ilustración 4.5.

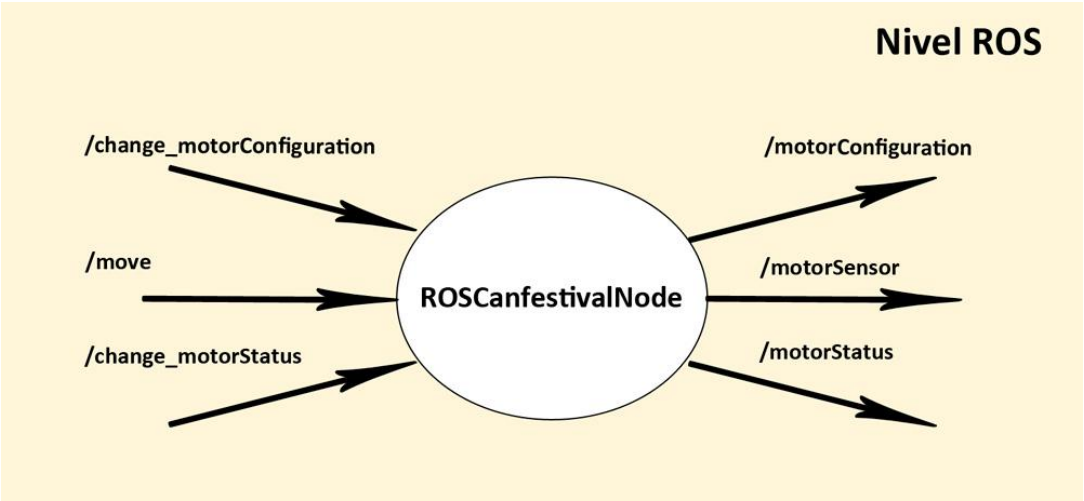


Ilustración 4.5. Arquitectura ROS de un controlador.
La ilustración muestra el nodo ROS implementado con los Topics a los que se suscribe para la recepción de las órdenes y los Topics en los que se publica la información.

Los diferentes partes que constituyen la red ROS creada son:

- **Nodos:**

  - **ROSCanfestivalNode**: Sus funciones han sido explicadas en el apartado 4.2.

- **Topics Suscritos:**

  - **/move:** En este topic se envía el mensaje de tipo *move* con el que se transmiten órdenes para mover el motor por posición o velocidad. El mensaje *move* está descrito en el anexo C.

  - **/change_motorStatus:** mediante de este topic se envía el mensaje tipo *motorStatus* que permite modificar el estado del motor, por ejemplo su habilitación e inhabilitación. El mensaje tipo *motorStatus* está descrito en el anexo C.

  - **/change_motorConfiguration:** A través de este topic se transmite el mensaje de tipo *motorConfiguration* que permite cambiar los parámetros de configuración del motor como su velocidad máxima, límites de posición, etc. El mensaje *motorConfiguration* está descrito en el anexo C.

- **Topics Publicados:**

  - **/motorStatus:** Se envía el mensaje de tipo *motorStatus* con los valores de estado del motor. Estos valores pueden ser utilizados para saber si han alcanzado sus límites de posición, sobre corriente, sobre temperatura, etc.

  - **/motorSensor:** los valores de posición, corriente y velocidad del motor son recibidos a través de este topic en un mensaje del tipo *motorSensor*. El mensaje *motorSensor* está descrito en el anexo C.

  - **/motorConfiguration:** A través de este topic se envían los valores actuales de configuración del motor mediante un mensaje del tipo *motorConfiguration*.

Para comprobar el correcto funcionamiento del controlador *ROSCanfestivalNode* se ha elaborado otro nodo ROS denominado ***Control*** para el envío de las diferentes órdenes que permiten cambiar la configuración, estado, posición y velocidad del motor. Este nodo *control* permite al usuario simular el envío de órdenes al controlador que podrían ser enviadas por una habilidad o aplicación. Adicionalmente se ha creado un tercer nodo llamado ***Listener*** que mostrará por pantalla los datos de funcionamiento recogidos del motor. La red ROS diseñada puede observarse en la Ilustración 4.6.

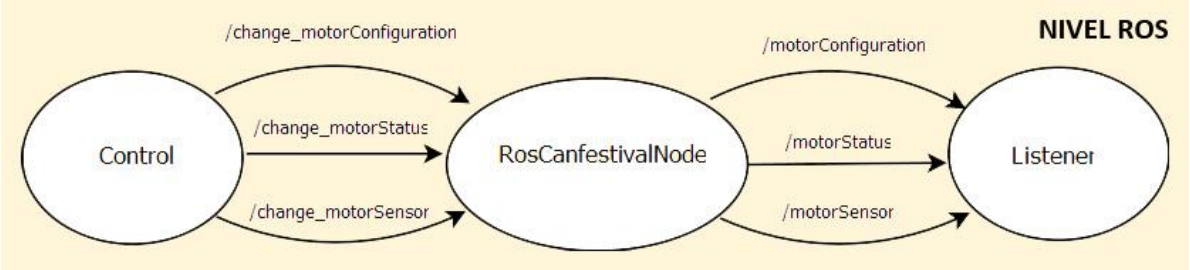


Ilustración 4.6. Red ROS creada para prueba de control de un motor.
Se muestra la arquitectura de control diseñada en ROS para la verificación del correcto funcionamiento del controlador. El nodo control envía órdenes mientras que el nodo *Listener* muestra por pantalla los datos recogidos del motor.

4.3.2 Red ROS para dos motores.

La base del robot móvil Mopi dispone de dos motores, por lo que es necesaria la ampliación de la red ROS básica diseñada para el uso de dos controladores CANopen, uno para cada motor. Antes de su instalación se realizará una prueba de funcionamiento previa a la instalación en el robot con la creación de dos nodos de control.

Siguiendo el diseño de la red ROS básica vista anteriormente en la Ilustración 4.6 se han creado dos nodos a partir de instancias de *ROSCanfestivalNode*. Estos nodos han sido renombrados como *rightWheels* y *leftWheels* y son los controladores CANopen de las ruedas del lado derecho e izquierdo respectivamente. También se han creado los nodos *control_right* y *control_left* para realizar las pruebas de funcionamiento. Estos nodos permiten al usuario el envío de las órdenes desde teclado. Para la monitorización de los datos de funcionamiento del motor se han creado los nodos *listener_right* y *listener_left*. Los topics utilizados en las comunicaciones siguen la misma estructura que los descritos en el apartado 4.3.1. Cada nodo controlador CANopen está suscrito a tres topics de entrada, por donde llegan las órdenes y publica en otros tres topics los valores recogidos del motor que tiene asignado. La nomenclatura de los topics es la misma a excepción de que su nombre comienza con el nombre del controlador CANopen que los utilice, es decir, las ordenes de configuración que se envían al nodo *leftWheels* utilizan el topic *leftWheels/change_motorConfiguration*. Esta red ROS de prueba, utilizada para el control ambos motores, se muestra en la Ilustración 4.7.

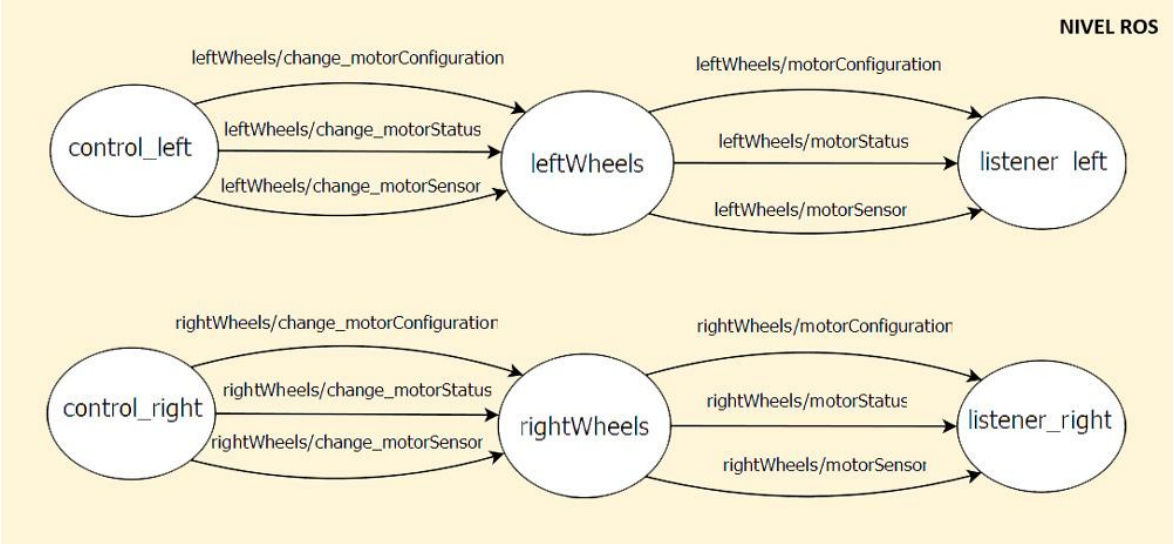


Ilustración 4.7: Red ROS para el control de dos motores.
Los nodos de control permiten el envío de órdenes a ambos motores y los datos de los motores son mostrados por pantalla mediante los nodos *listener*.

4.3.3 Instalación y verificación de funcionamiento en el robot Mopi

Una vez comprobado el correcto funcionamiento de ambos controladores, el siguiente paso es su integración en el robot Mopi y la verificación del correcto funcionamiento. Para este propósito se ha decidido utilizar un paquete de ROS llamado *JOY* que envía, a través de un topic del mismo nombre, los botones que han sido activados en un mando inalámbrico o Joypad genérico. Para asignar diferentes funciones a los botones, permitiendo el manejo del robot, ha sido necesario implementar un nodo que determine que órdenes de funcionamiento deben enviarse a los controladores dependiendo del botón que haya sido pulsado, este nodo ha sido denominado ***JoyMopiController.*** Por tanto, para esta prueba, en lugar de utilizar los nodos de control para el envío de órdenes, se utilizará el Joypad permitiendo el control del robot. La red ROS diseñada se observa en la Ilustración 4.8.

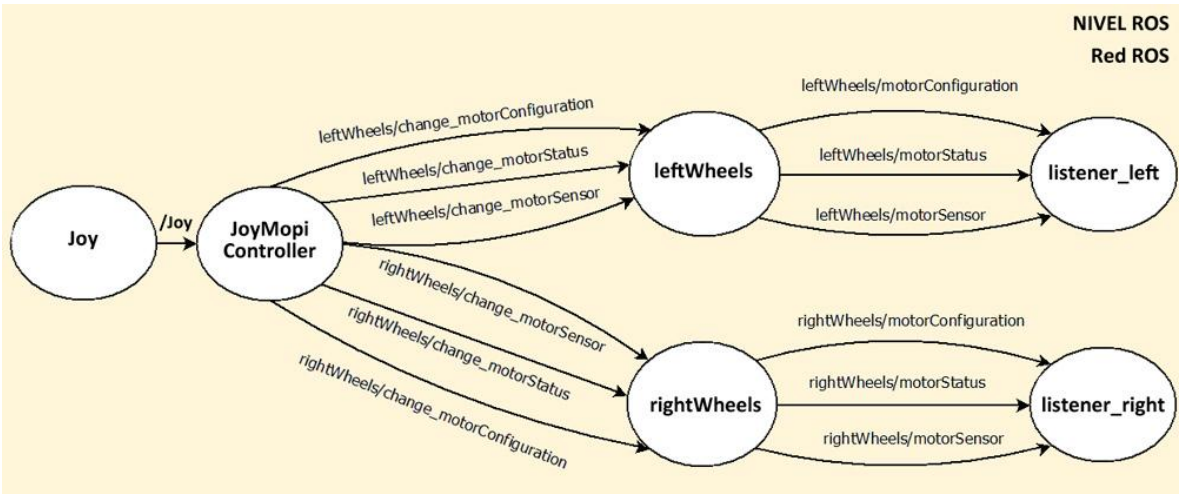


Ilustración 4.8. Red Ros de operatividad del robot Mopi.
Esta red permite la operatividad del robot Mopi mediante el uso de un mando inalámbrico o Joypad.

El diagrama de casos de uso de ***JoyMopiController*** se observa en la Ilustración 4.9. *JoyMopiController* está suscrito al topic *joy*, a través del cual recibe que botones han sido pulsados en el joypad y pública las órdenes pertinentes en los topics a los que están suscritos ambos nodos controladores *leftWheels* y *rightWheels*. En concreto las funciones implementadas han sido diversos movimientos controlados por velocidad así como la activación y desactivación de motores.

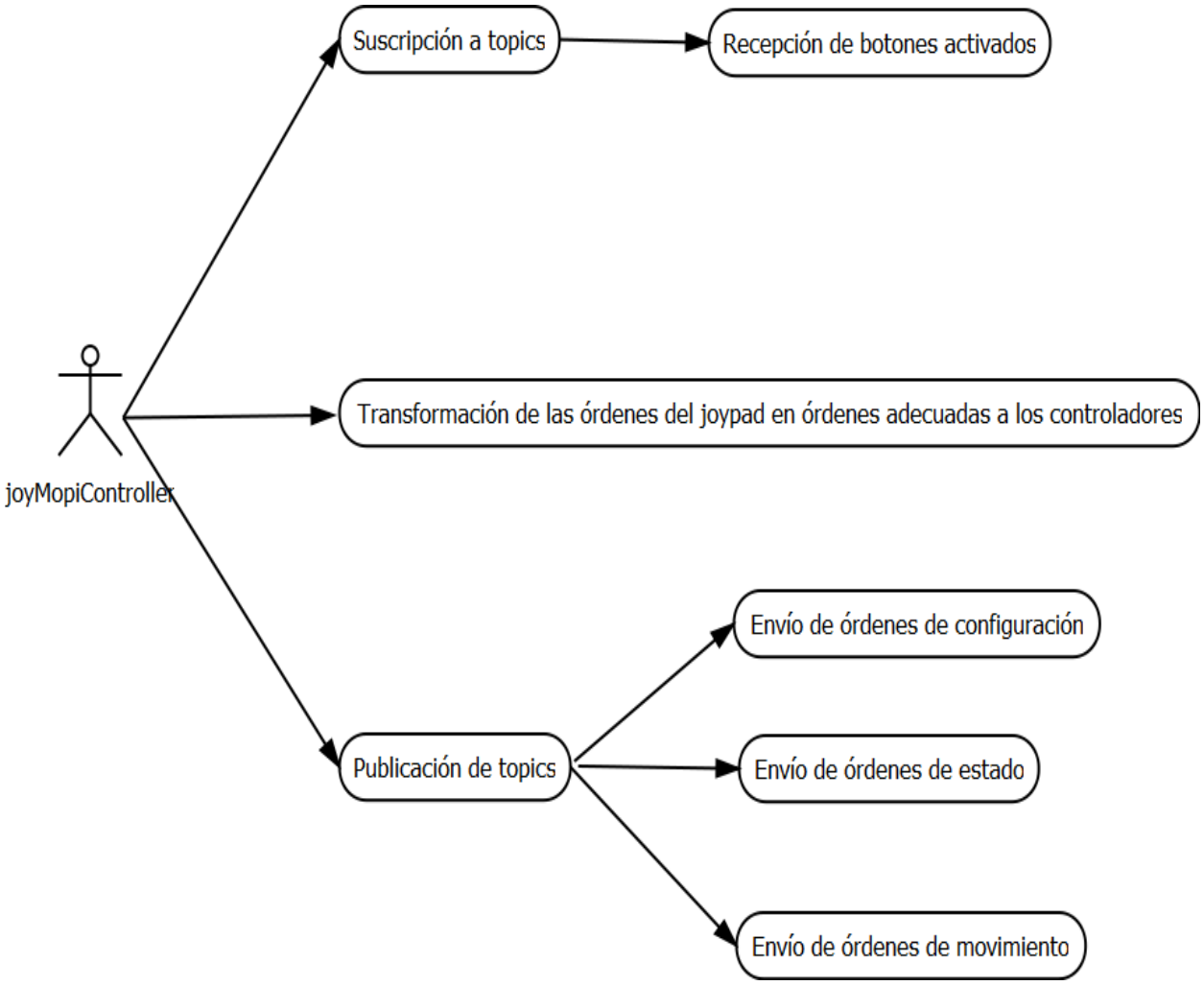


Ilustración 4.9. Diagrama de casos joyMopiController.
El nodo *joyMopiController* permite la operatividad del robot Mopi mediante un joypad. Permite asignar órdenes a los diferentes botones del joypad.

4.4 Nivel Canfestival

En el nivel Canfestival se crea una red CANopen En este apartado se describe el funcionamiento y configuración de esta red. Una red CANopen consta de un nodo maestro y uno o varios nodos esclavos. También es posible la utilización de más de un nodo maestro. Esta opción multimaestro permite que, en caso de colapso de uno de los maestros, puedan seguir funcionando los demás motores. Puede pensarse que en el caso de la base del robot Mopi, que consta de dos motores, no sirve el funcionamiento de un

único motor, pero hay que recordar que el objetivo principal de este proyecto ha sido el desarrollo de un controlador CANopen que permita el control de cualquier motor. Por ello, este controlador podría instalarse por ejemplo en los brazos de un robot de manera que en caso de colapso del maestro de uno de los brazos el otro podría seguir moviéndose independientemente. Por esta razón se ha decidido utilizar varios maestros de forma que cada nodo esclavo ejecute solamente las órdenes del nodo maestro que tenga asignado.

Para el control de un motor en la red CANopen creada es necesario la existencia de un nodo maestro, creado por el controlador *ROSCanfestivalNode*, y un nodo esclavo, representado por el dispositivo MCDC3006C y encargado de la ejecución de las órdenes de las órdenes de funcionamiento recibidas desde el maestro.

A continuación se describe la red CANopen utilizada para el control de los motores de la base del robot. La base del robot Mopi consta de dos motores, por lo que esta red constará de dos nodos maestros y dos nodos esclavos. Debe recordarse que en el nivel ROS se crearon dos controladores a partir de instancias de *ROSCanfestivalNode* y fueron renombrados como *rightWheels* y *lefftWheels (ver apartado 4.3.2)*. Por tanto, los controladores *rightWheels* y *lefftWheels*, además de constituir cada uno un nodo de la red ROS, representan dos nodos maestros en la red CANopen. Para la ejecución de las órdenes se utilizan dos dispositivos MCDC3006C, uno por cada motor. Cada dispositivo de control MCDC3006C es un nodo esclavo que deberá obedecer las órdenes del maestro que tienen asignado.

La Ilustración 4.10 muestra un esquema de la red CANopen implementada. Como se puede observar cada nodo de la red, maestro y esclavo, tiene su propio diccionario de objetos que configura sus comunicaciones y funcionalidades. El firmware de los nodos esclavos, dispositivos MCDC3006C, incorpora un diccionario por defecto donde se configuran automáticamente sus comunicaciones según el valor del identificador que se asigne al dispositivo como se mostró en el apartado 2.4.4.1. Los diccionarios de los nodos maestros se han configurado de manera que permitan comunicarse con los nodos esclavos. La configuración de las comunicaciones de la red CANopen se explica en el apartado 4.4.1.

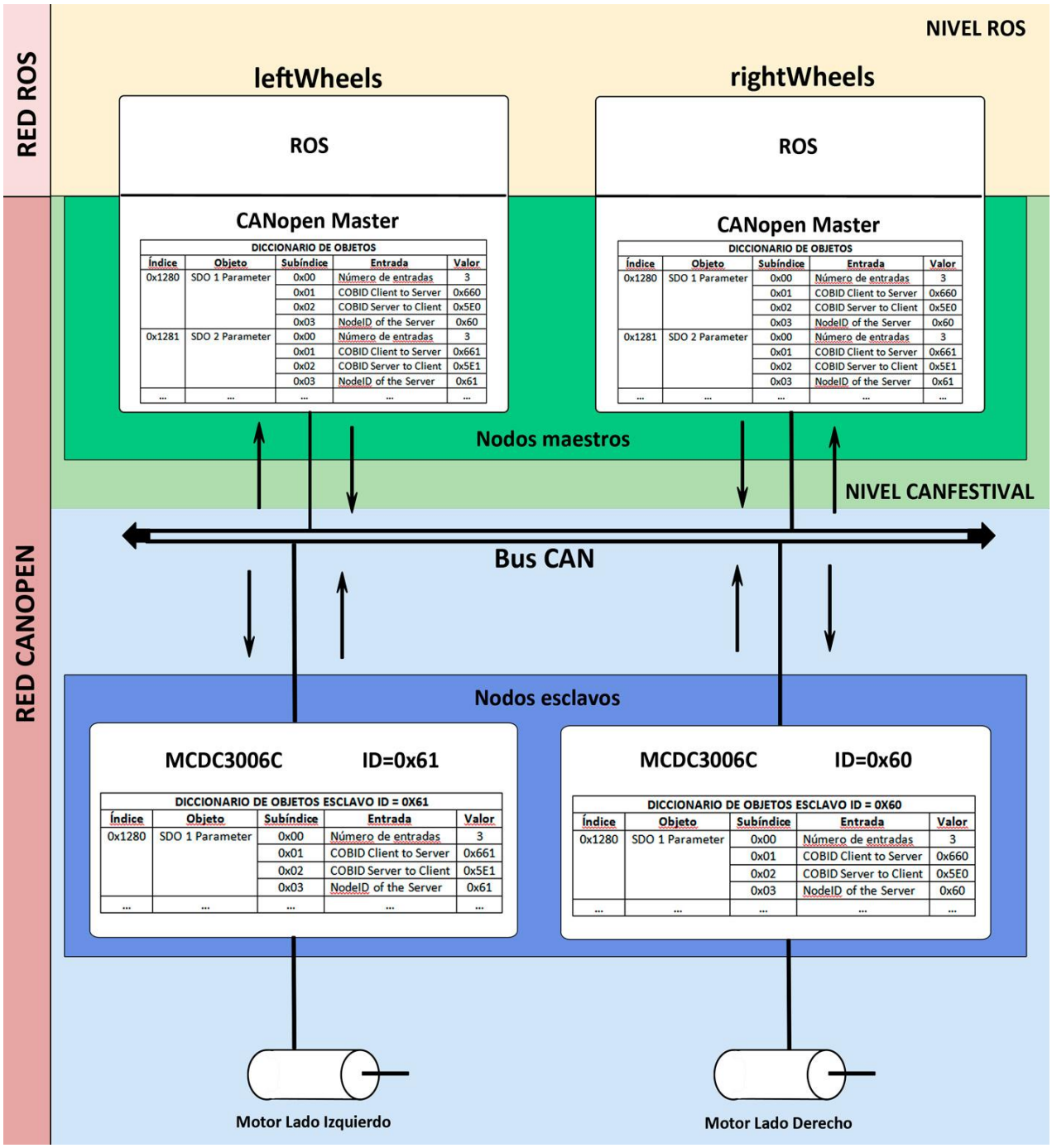


Ilustración 4.10. Esquema de la Red CANopen.

Existe un nodo maestro para controlar cada nodo esclavo. La ilustración muestra el esquema de la red CANopen utilizada para el control del robot Mopi. Consta de dos nodos maestros que controlan los nodos esclavos asignados.

4.4.1 Configuración de la red CANopen.

Como se explicó en el apartado 2.4.4.1, CANopen proporciona una configuración por defecto de los identificadores de los mensajes o COBIDs (Communication Object Identifier). Los diferentes COBIDs que utilizaran los nodos para su comunicación se encuentran declarados en sus diccionarios de objetos.

Los mensajes SDO y PDO pueden ser de recepción o de transmisión. Para que exista comunicación de un maestro con un nodo esclavo, el COBID del mensaje de transmisión del nodo maestro debe coincidir con el COBID del mensaje de recepción del nodo esclavo y viceversa. El ejemplo de la Ilustración 4.11 muestra una configuración en la que existe un nodo esclavo, con la configuración COBID por defecto e identificador ID=0x01, y un nodo maestro. Dependiendo del número de identificador del esclavo se asignarán unos COBIDs por defecto. Será necesario crear el diccionario de objetos del maestro con los COBIDs del nodo esclavo para poder comunicarse.

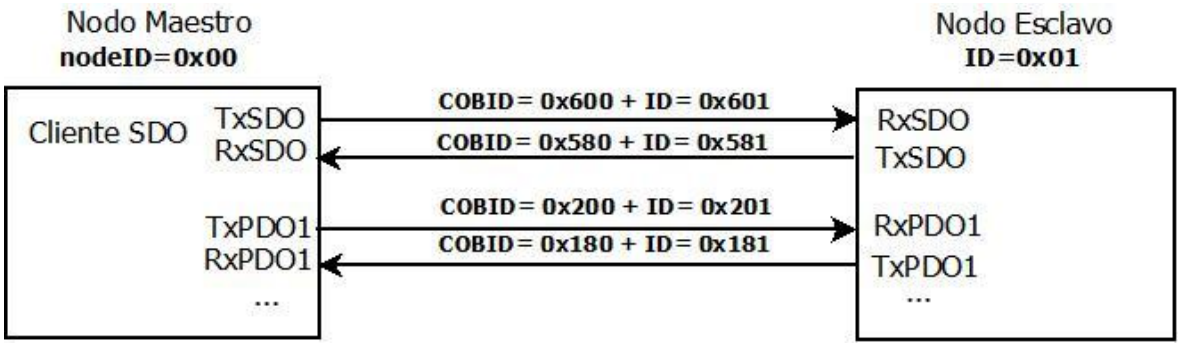


Ilustración 4.11. Ejemplo de configuración de una red CANopen.
Se muestran los diferentes COBIDs utilizados para establecer la comunicación entre un nodo maestro y un esclavo.

La configuración de la red CANopen utilizada por el robot consta de dos nodos maestros y dos nodos esclavos constituidos por los dispositivos MCDC3006C. Estos dispositivos incorporan la configuración por defecto en la que se establecen los COBIDs, para un mensaje de transmisión y otro de recepción tipo SDO así como tres mensajes de transmisión y tres de recepción del tipo PDO, según el identificador de nodo que posean en la red CANopen (ver apartado 3.2.1). Por tanto los COBIDs que serán utilizados para la comunicación entre los maestros y esclavos se observan en la Ilustración 4.12.

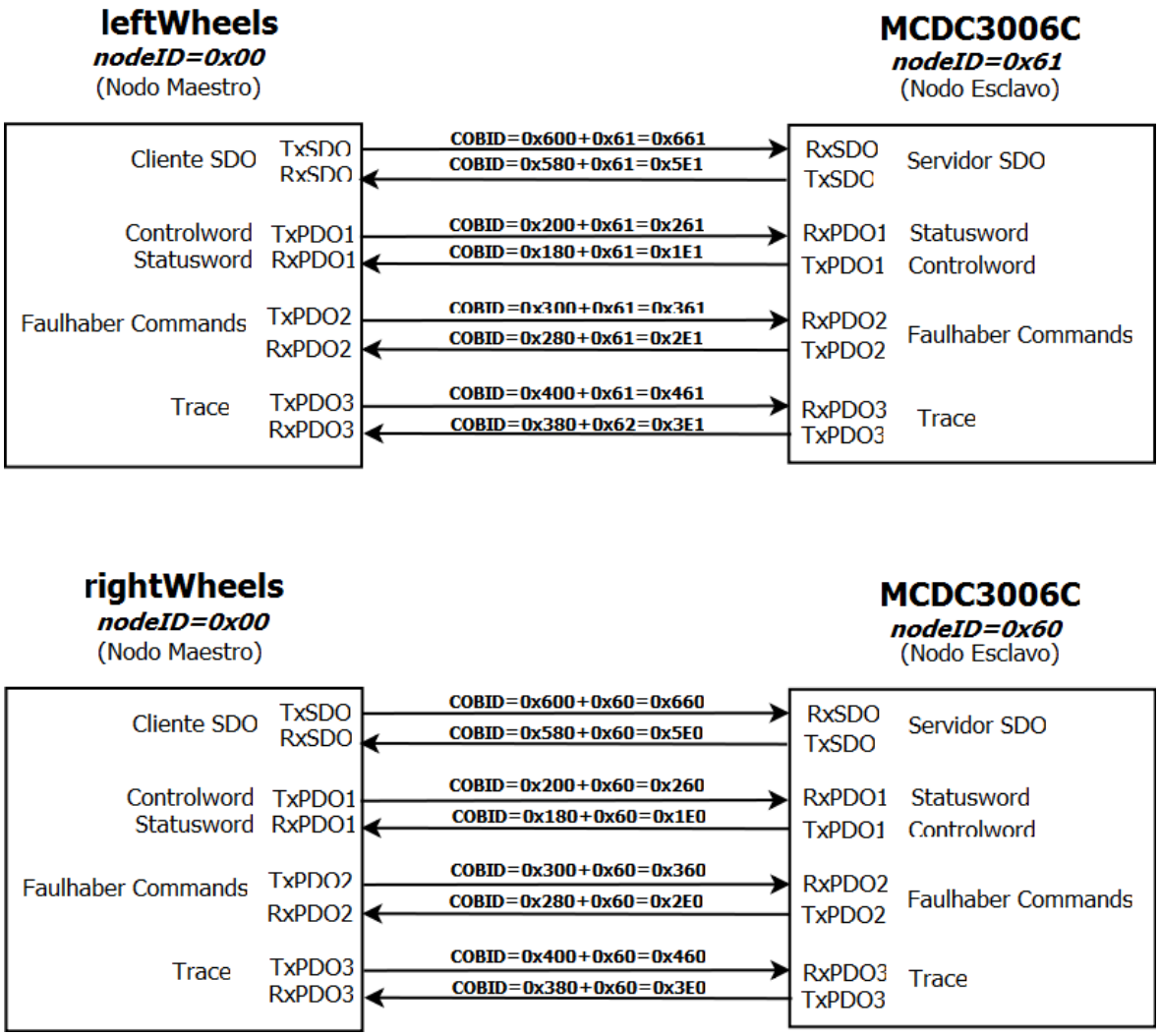


Ilustración 4.12.Configuración COBID de la red CANopen del robot Mopi.
Se muestran los identificadores de mensajes utilizados para que cada maestro se comunique con su nodo esclavo asignado.

Desde el principio de este proyecto se ha buscado la simplicidad en la arquitectura diseñada con el objetivo de facilitar la adición de nuevos motores a la red CANopen. Cada nodo maestro se crea a partir de la instancia del mismo diseño software y todos ellos comparten el mismo diccionario de objetos. Este diccionario de objetos contiene todos los COBIDs utilizados en las comunicaciones de la red CANopen. Cada nodo maestro utilizará solo los COBIDs que le permitan establecer la comunicación con el nodo esclavo que tenga asignado. El diccionario de los nodos maestros puede observarse en el anexo A.

4.5 Ejemplo de funcionamiento del robot Mopi.

Después de explicar cada uno de los niveles de la arquitectura por separado, nivel ROS y nivel Canfestival, a continuación se describe el funcionamiento desde una visión global. En concreto se describe el envío de una orden al robot mediante Joypad así como proceso de adquisición de datos procedentes de los motores. En el ejemplo de la Ilustración 4.13 se envía una orden al robot, que se encuentra detenido, para moverlo hacia delante. Para

moverse hacia delante el robot debe mover los motores en sentido contrario debido a que en el robot se encuentran instalados en sentidos opuestos. En concreto el motor de las ruedas del lado izquierdo se mueve en sentido anti horario y el de las ruedas del lado derecho en sentido horario.

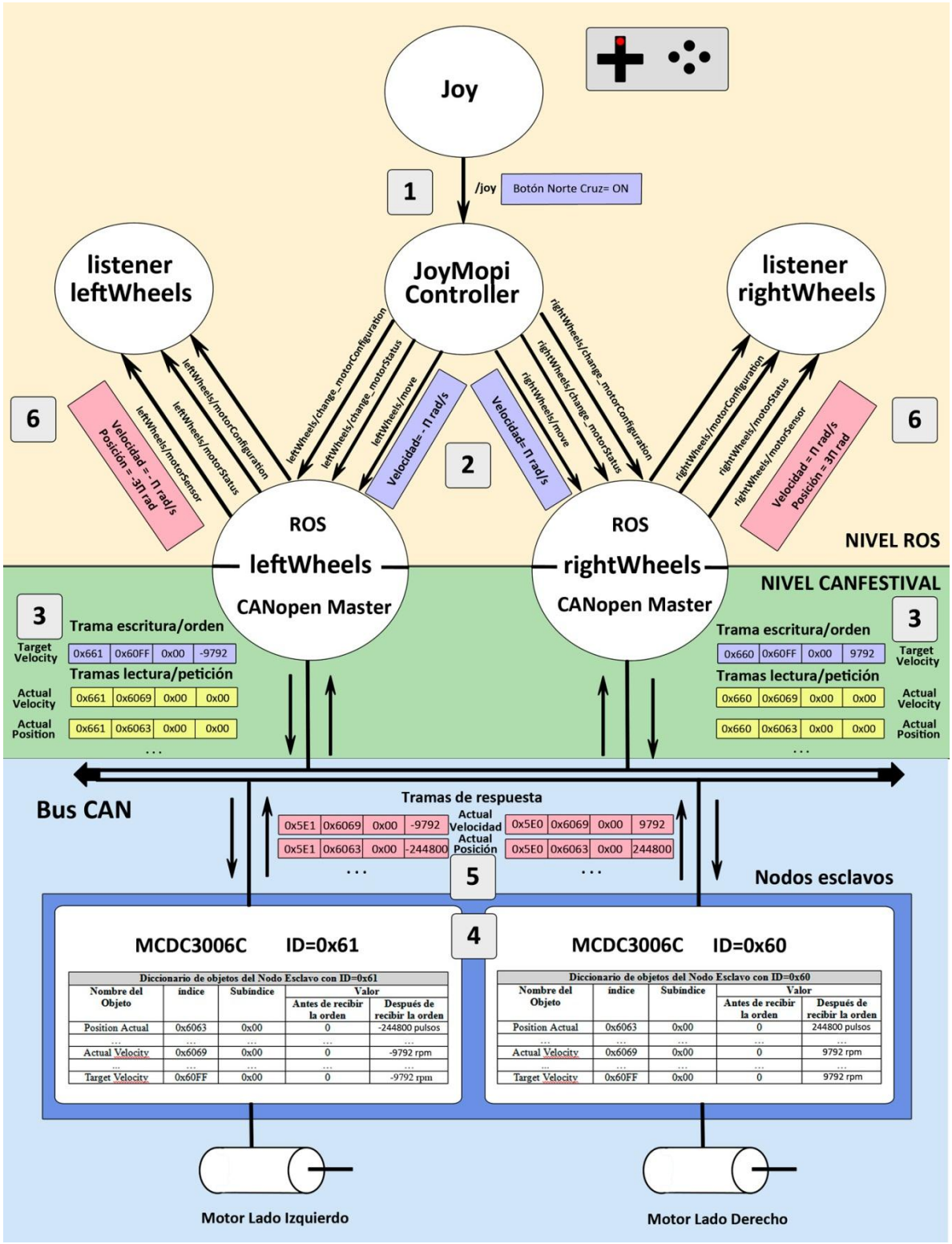


Ilustración 4.13. Ejemplo funcionamiento de la arquitectura software implementada.
La ilustración muestra el proceso de envío de una orden de velocidad y el bucle de petición de datos de funcionamiento de los motores

A continuación se describen cada uno de los pasos numerados de la Ilustración 4.13 que permiten que el robot se mueva hacia delante:

1. El nodo ***Joy*** envía a través del topic */joy* el botón que ha sido pulsado. En este caso se ha pulsado el botón norte de la cruz del joypad.

2. El nodo ***JoyMopiController*** permite la operatividad del robot Mopi a través de joypad. Para ello está suscrito al topic */joy* a través del cual recibe que botones del joypad han sido activados. Dependiendo de los botones activados se envían las órdenes pertinentes a los nodos *righWheels* y *leftWheels*. Este ejemplo expone el envío de la orden que permite a robot moverse hacia delante. Para ello se publicará en el topic */leftWheels/move* el valor velocidad = -π rad/s y en el topic */rightWheels/move* el valor velocidad= π rad/s. Estos valores son las velocidades de la rueda del robot.

3. Los nodos *rightWheels* y *leftWheels* tienen una doble función. Se encargan de la transformación de órdenes en tramas CAN y la recopilación de datos procedentes de los motores. Estos nodos se encuentran constantemente enviando tramas de petición de datos a los nodos esclavos. Cuando se termina el bucle de petición se comprueba si se han recibido órdenes y de ser así se envía la trama CANopen correspondiente. En este ejemplo se desea mover el robot con una velocidad de 10π rad/s. Para alcanzar este valor de velocidad es necesario tener en cuenta la reductora que existe entre la rueda y el eje del motor que permite aumentar el par de los motores. Los nodos *rightWheel* y *leftWheels* realizan la conversión necesaria para alcanzar la velocidad indicada. También deben realizar una conversión de rad/s a rpm (revoluciones por minuto) puesto que las unidades por defecto de los dispositivos de control CANopen son rpm. En este ejemplo para mover las ruedas a una velocidad de π rad/s será necesario enviar un valor de 9792 rpm a los motores. Una vez enviada la orden comienza de nuevo el bucle de petición de datos.

4. Los motores se encuentran activados y detenidos en la posición inicial. Una vez llegan las ordenes de velocidad a los dispositivos de control MCDC3006C se escribe el valor de dicha velocidad en el objeto *Target Velocity* de sus diccionarios de objetos indicando las velocidades que deben alcanzar los motores. En este momento los motores comienzan a moverse y el dispositivo MCDC3006C sobrescribe los valores de los objetos *Actual Position* y *Actual Velocity* de su diccionario con los valores de posición y velocidad obtenidos del encoder del motor.

5. Se han recibido tramas de petición de datos provenientes de los nodos maestros, en concreto este ejemplo se muestran las tramas que solicitan los valores de velocidad y posición del motor. Dichas tramas han llegado después de la orden de movimiento por lo que el nodo esclavo, el dispositivo MCDC3006C, contesta con los valores de velocidad y posición que se encuentran en los objetos del diccionario *Actual Velocity* y *Actual Position* respectivamente en el momento de la

recepción de la petición. En este ejemplo se ha supuesto que las tramas han llegado en el momento en que los motores han dado una vuelta y media (3π rad) o lo que es lo mismo, para este caso teniendo en cuanta la reductora, 244.800 pulsos que se han detectado en su encoder.

6. Los nodos *rightWheels* y *leftWheels* reciben las contestaciones a sus peticiones de datos y las preparan para la salida a través del topic oportuno. Se han realizado las conversiones necesarias de rpm a rad/s y de pulsos a radianes. Los nodos *listener* mostrarán por pantalla que las ruedas izquierdas se han desplazado -3 π rad y se encuentran desplazándose a una velocidad de - π rad/s mientras que las ruedas de lado derecho se han desplazado 3 π rad y se están moviendo a una velocidad de π rad/s.

4.6 Descripción de la implementación de la arquitectura software

En este apartado se describe la implementación del controlador CANopen. En la Ilustración 4.14 se observa la arquitectura software en la que el nivel ROS de *ROSCanfestivalNode* utiliza la librería ***canopen_driver*** que ha sido implementada para este proyecto. En la implementación de esta librería se ha buscado la modularidad para permitir la reutilización del código en el desarrollo de nuevos controladores.

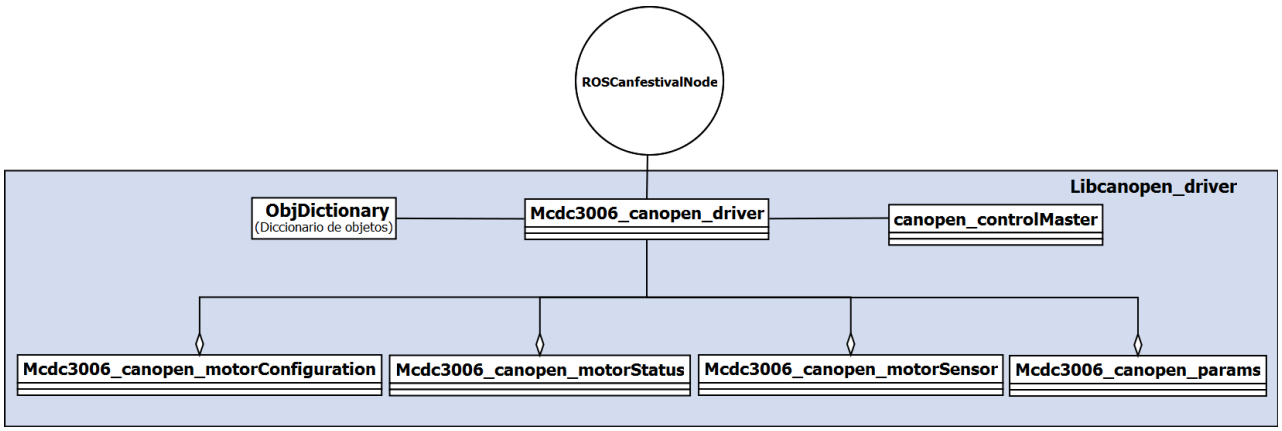


Ilustración 4.14. Diagrama de clases de la arquitectura software.
Existen dos clases de control denominadas Mcdc3006_canopen_driver y *canopen_controlMaster*. Además, existen otras cuatro clases de datos. *ObjDictionary* almacena el diccionario de objetos utilizado por *ROSCanfestivalNode* para constituir un nodo maestro en la red CANopen.

La librería *canopen_driver* está compuesta por dos clases de control:

- **Canopen_controlMaster:** Consiste en una clase de control de las comunicaciones CANopen. Inicializa las comunicaciones incluyendo el maestro CANopen así como su finalización.

- **Mcdc3006_canopen_driver:** En esta clase se encuentran implementadas todas funcionalidades de los motores para permitir su control. Permite la abstracción del protocolo de comunicación CANopen. Realiza el envío de las tramas de órdenes y

peticiones así como la lectura de las tramas entrantes. Como se observa en la Ilustración 4.14 el nivel ROS de *ROSCanfestivalNode* se comunica directamente con esta clase para establecer las comunicaciones CANopen con los nodos esclavos. Para establecer las comunicaciones la clase *Mcdc3006_canopen_driver* utiliza la clase de control *Canopen_controlMaster* para la inicialización y finalización del puerto de comunicaciones así como del maestro CANopen. Inicializadas las comunicaciones, la clase *Mcdc3006_canopen_driver* utiliza el diccionario de objetos *ObjDictionary,* común para todos los nodos maestros, el cual incorpora todos los identificadores de mensajes o COBIDs de la red CANopen. La clase *Mcdc3006_canopen_params* permite utilizar solo los COBIDs de los mensajes PDO que pertenecen al nodo esclavo asignado.

Establecidas la comunicación con el nodo esclavo se comienzan a recibir tramas con los datos de los motores. Estos datos son almacenados en tres clases dependiendo de su funcionalidad*:*

- *Mcdc3006_canopen_motorConfiguration*
- *Mcdc3006_canopen_motorSensor*
- *Mcdc3006_canopen_motorStatus*

Por ejemplo, se envía el valor de la nueva posición que debe alcanzar el motor, posteriormente se envía la trama de petición con el número de identificación del nodo esclavo y la dirección de la entrada del diccionario de objetos donde se encuentra el valor de la posición actual. Se trata de una comunicación asíncrona por lo que se espera la contestación con el valor. Cuando se recibe el valor de la posición es almacenado en la clase *Mcdc3006_canopen_motorSensor.*

Como se ha comentado, la clase de control *Mcdc3006_canopen_driver* utiliza una serie de clases de datos para agrupar de los diferentes datos de configuración, estado, y movimiento de los motores así como los parámetros que determinan que identificadores de mensaje (COBIDs) del diccionario de objetos debe usar para comunicarse con el nodo esclavo asignado.

- **Mcdc3006_canopen_motorConfiguration**. Almacena todos los parámetros relacionados con la configuración del motor como la velocidad máxima, límites de posición, nombre del controlador, etc. Además se suministran funciones para la lectura y modificación de estos datos por parte de la clase de control Mcdc3006_canopen_driver.

- **Mcdc3006_canopen_motorSensor:** En esta clase se almacenan los datos de posición, velocidad y corriente consumida por el motor. También se suministran funciones para la lectura y modificación de dichos datos por parte de la clase de control Mcdc3006_canopen_driver.

- **Mcdc3006_canopen_motorStatus:** Esta clase es utilizada para el almacenamiento de datos relacionados con el estado del actuador. Como en las otras clases se

suministran funciones para modificar dichos parámetros y permitir el acceso de la clase de control Mcdc3006_canopen_driver.

- **Mcdc3006_canopen_params:** Al utilizar un solo diccionario que es compartido por todos los nodos maestros es necesaria la utilización una serie de parámetros que permitan a cada nodo saber que identificadores de mensajes de tipo PDO deben utilizar para comunicarse con el nodo esclavo que tiene asignado. Estos parámetros son asignados mediante los archivos de configuración XML. La Tabla 4.1 muestra la numeración, según su posición en el diccionario de objetos, que debe indicarse para el para el envío de eventos PDO. Por ejemplo a través del envío PDO1 puede establecerse la inhabilitación y habilitación de los motores. Si se quiere habilitar el motor de las ruedas derechas debe indicarse que el TxPDO1 a enviar es el número 0, en caso de querer habilitar el motor izquierdo se indicará que es el TxPDO1 número 3.

Nº TxPDO	**Índice**	**Tipo de PDO**	**Objeto**	***Nombre del nodo***
0	0x1800	TxPDO1	Controlword	rightWheels
1	0x1801	TxPDO2 Faulhaber	FaulhaberCommand	rightWheels
2	0x1802	TxPDO3 Trace	TraceConfiguration	rightWheels
3	0x1803	TxPDO1	Controlword	leftWheels
4	0x1804	TxPDO2 Faulhaber	FaulhaberCommand	leftWheels
5	0x1805	TxPDO3 Trace	TraceConfiguration	leftWheels

Tabla 4.1 Numeración TxPDOs.
La tabla muestra la numeración utilizada para el envío de los mensajes PDO mediante la API de Canfestival.

Cuando se realiza una petición mediante un mensaje PDO debe indicarse el índice del mensaje tipo RxPDO que recibirá el objeto. Por ejemplo si se desea recibir el estado del motor derecho (rightWheels), el cual se transmite mediante el objeto *Statusword*, se indica el índice 0x1400 del mensaje RxPDO1. Ver Tabla 4.2.

Índice	**Tipo de PDO**	**Objeto**	**Nombre del nodo**
0x1400	RxPDO1	Statusword	rightWheels
0x1401	RxPDO2 Faulhaber	FaulhaberData	rightWheels
0x1402	RxPDO3 Trace	TraceData	rightWheels
0x1403	RxPDO1	Statusword	leftWheels
0x1404	RxPDO2 Faulhaber	FaulhaberData	leftWheels
0x1405	RxPDO3 Trace	TraceData	leftWheels

Tabla 4.2 Índices de los mensajes RxPDO.
La tabla muestra los índices utilizados para el envío de peticiones mediante mensajes PDO.

Cada nodo de la red CANopen dispone de un diccionario de objetos que contiene los diferentes objetos que definen los tipos datos, permiten la comunicación, definen su funcionalidad, etc. El diccionario de objetos común creado para los maestros es el siguiente:

- **ObjDictionary (diccionario de objetos):** *ObjDictionary* es una tabla de variables en la que se ordenan los diferentes objetos de comunicación que permiten a los maestros comunicarse con los nodos esclavos asignados. Entre estos objetos se pueden encontrar aquellos que definen los identificadores de mensajes o COBIDs utilizados en la red CANopen. Se puede observar el diccionario de objetos de los maestros en el anexo A. Por diseño se ha decidido utilizar un único diccionario de objetos del que cada nodo maestro tiene una copia. Esto facilita la adición de un nuevo nodo a la red o cambios de configuración.

  Para la creación del diccionario de los maestros la capa de aplicación de Canfestival facilita una herramienta de edición. El proceso de construcción de un diccionario de objetos mediante esta herramienta puede encontrarse en el anexo B.

Un sistema robótico normalmente está compuesto por una serie de motores de diferentes características y con diferentes funciones, es decir, no todos los motores van a tener los mismos límites de posición o la misma velocidad máxima por ejemplo. Además, un mismo motor puede ser configurado, por ejemplo, con límites y velocidades diferentes dependiendo del cual vaya a ser su función. Por todo esto es importante poder establecer una configuración inicial para evitar problemas de funcionamiento. Esta configuración se realiza mediantes archivos XML que son facilitados al nodo ejecutable *ROSCanfestivalNode*.

- **Archivo configuración XML:** Los archivos XML contienen los diferentes parámetros de configuración que son facilitados al controlador *ROSCanfestivalNode* y enviados nada más inicializar las comunicaciones CANopen a los dispositivos MCDC3006C para el correcto funcionamiento de los motores. Cada controlador *ROSCanfestivalNode* tiene su propio archivo XML que envía al nodo esclavo asignado. En el anexo D puede consultarse un ejemplo de un archivo XML para conocer mejor su composición.

  Estos archivos de configuración disponen de dos tipos de datos. Parte de ellos serán utilizados por *ROSCanfestivalNode* para crear las comunicaciones de la red ROS así como la creación del nodo maestro de la red CANopen para comunicarse con el nodo esclavo asignado. Los demás parámetros suministrados por los archivos XML configuran el funcionamiento de los motores.

Los parámetros de configuración utilizados por *ROSCanfestivalNode* son:

- Nombre del controlador.
- ID del dispositivo MCDC3006C a controlar.
- Velocidad de transmisión del bus CANopen.
- Tipo de driver.
- Numeración de los mensajes TxPDO e índices de los mensajes RxPDO que utilizará para comunicarse con el nodo esclavo asignado.

Los parámetros que *ROSCanfestivalNode* transmitirá al motor para su configuración son:

- Máxima posición.
- Mínima posición.
- Máxima velocidad.
- Calibración del sensor de posición.
- Máximo voltaje.
- Mínimo voltaje.
- Número de pulsos por revolución.
- Factores de Multiplicación.
- Factor de reducción.
- Corriente máxima.
- Máxima temperatura del dispositivo de control.

4.7 Incorporación de un nuevo motor.

La arquitectura software diseñada facilita la adición de un nuevo controlador CANopen. Como se explicó en el apartado 4.1 la comunicación se estructura en dos niveles: el nivel ROS y el nivel Canfestival en los que se crean las redes ROS y CANopen respectivamente.

Configuración de la red ROS:

Los diferentes nodos controladores se crean a partir de instancias de *ROSCanfestivalNode*. Para crear los nuevos nodos es necesario crear un archivo de configuración XML que debe contener al menos el nombre del nuevo nodo y el número del identificador del nodo esclavo que tendrá asignado. Para este ejemplo, el nombre de este nuevo controlador CANopen es *nuevoNodo* y el identificador de su nodo esclavo es ID=0x62 siguiendo con la numeración de los motores del robot Mopi. Los diferentes topics se crean automáticamente a partir del nombre del nuevo controlador como se muestra en la Ilustración 4.15.

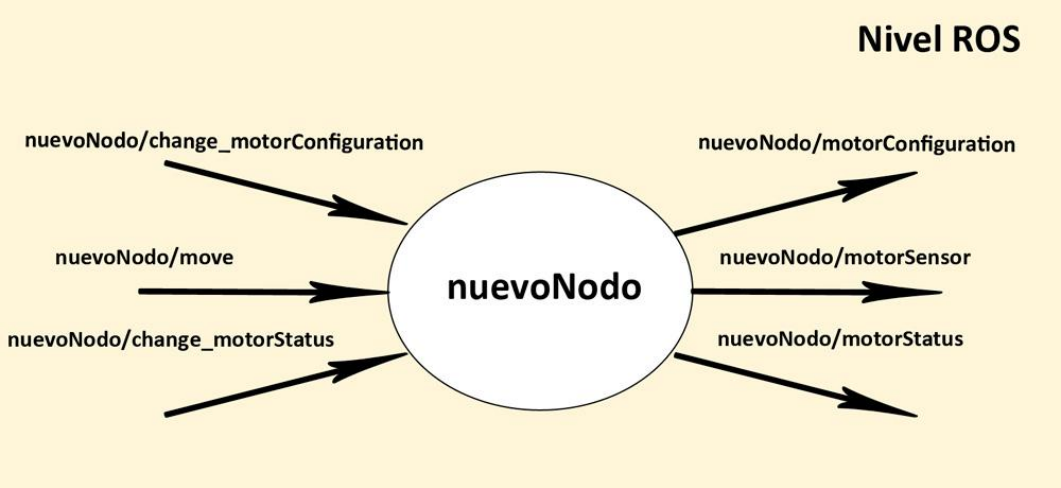


Ilustración 4.15. Introducción de un nuevo nodo en la red ROS.
Se muestra la creación de un nodo llamado *nuevoNodo* en la red ROS junto con la nomenclatura de los Topics de entrada y salida.

Configuración de la red CANopen:

Una de las principales ventajas del protocolo CANopen es su facilidad de configuración al incorporar un nuevo nodo a la red. Esta ventaja junto al diseño de la arquitectura que se ha desarrollado facilita la conexión de un nuevo dispositivo de control MCDC3006C u otro dispositivo de control que utilice el protocolo CANopen. A continuación se describe el proceso de configuración de un nuevo motor que utilizará un dispositivo de control MCDC3006C como nodo esclavo de la red CANopen.

1. **Asignación del número de identificación al dispositivo de control.**

   Lo primero que se debe hacer es asignar el número de identificación con el que el dispositivo de control se identificará en la red CANopen. Para este ejemplo se establece el número *ID=0x62* con el propósito de seguir la numeración establecida en este proyecto. Para ello es necesaria la utilización de la aplicación desarrollada por el fabricante del dispositivo como se describe en el anexo G.

2. **Ampliación del diccionario de objetos del maestro con los nuevos COBID.**

   Como se explicó en el apartado 2.4.4.1 el diccionario de objetos de los esclavos dispone de una configuración de comunicación por defecto que establece los diferentes COBIDs o identificadores de mensajes según el identificador del nodo. Del mismo modo que para los nodos ya instalados, la Ilustración 4.16 muestra los COBIDs utilizados por el nuevo nodo MCDC3006C.

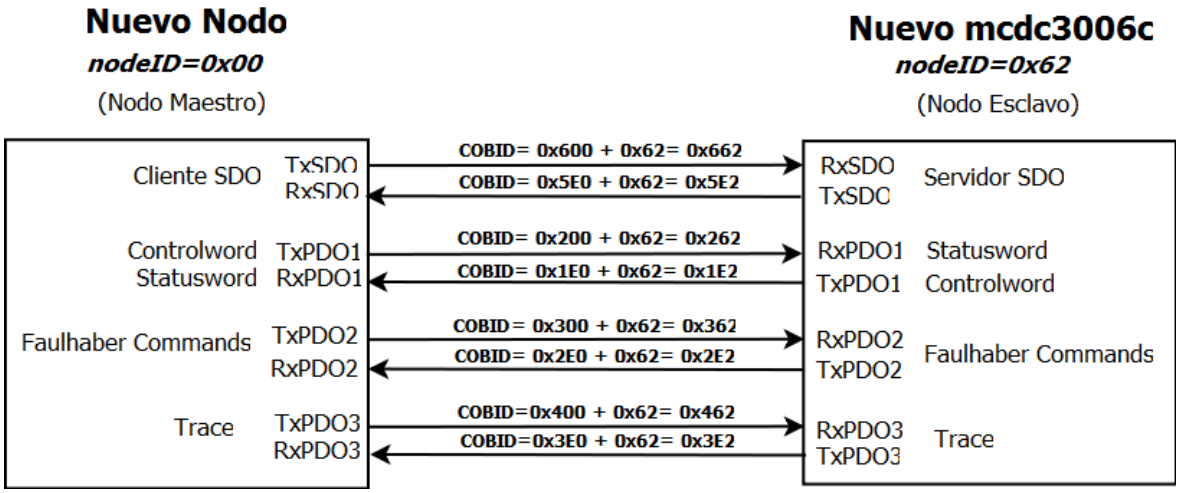


Ilustración 4.16. Identificadores de mensaje para el nuevo nodo MCDC3006C.
Identificadores de mensajes utilizados para la comunicación del maestro con el nuevo nodo esclavo.

Debe tenerse en cuenta que las especificaciones para los dispositivos de control que se encuentran en el documento DS402 de la CiA [9] solo definen el uso de los mensajes de transmisión TxPDO1 y de recepción RxPDO1 que deben transportar los objetos *Controlword* y *Statusword* respectivamente. Por tanto es posible que existan dispositivos de control de otros fabricantes que no se utilicen más mensajes PDO y no sea necesario todos los COBIDs o que por el contrario facilite más mensajes PDO y sea necesario añadir más COBIDs al diccionario de objetos. En caso de que los mensajes PDO adicionales transporten objetos diferentes es necesario especificarlo en el diccionario.

Una vez se tiene claro los nuevos COBIDs, deben ser añadidos al diccionario de objetos común utilizado por los maestros. Para ello debe utilizarse la herramienta de creación de diccionarios de objetos proporcionada por Canfestival. Los pasos a seguir en la implementación del diccionario de objetos mediante esta herramienta se encuentran descritos en el anexo B.

3. Nuevo archivo de configuración XML.

Puesto que cada motor puede tener una funcionalidad diferente es necesaria la creación de un nuevo archivo de configuración para establecer el comportamiento del nuevo motor. También deben indicarse que COBIDs de los mensajes TxPDO y RxPDO, del diccionario de objetos común, utiliza el nodo maestro para comunicarse con el nodo esclavo asignado. Continuando con la numeración seguida en este proyecto, la posición de los mensajes TxPDO del nuevo nodo se observa en la Tabla 4.3. En el caso de los mensajes RxPDO se observan sus índices en la Tabla 4.4. En el anexo D se encuentra un ejemplo de un archivo de configuración XML.

Nº TxPDO	Índice	Tipo de PDO	Objeto que transporta	*Nombre del nodo esclavo*
0	0x1800	TxPDO1	Controlword	rightWheels
1	0x1801	TxPDO2 Faulhaber	FaulhaberCommand	rightWheels
2	0x1802	TxPDO3 Trace	TraceConfiguration	rightWheels
3	0x1803	TxPDO1	Controlword	leftWheels
4	0x1804	TxPDO2 Faulhaber	FaulhaberCommand	leftWheels
5	0x1805	TxPDO3 Trace	TraceConfiguration	leftWheels
6	0x1806	TxPDO1	Controlword	Nuevo Nodo
7	0x1807	TxPDO2 Faulhaber	FaulhaberCommand	Nuevo Nodo
8	0x1808	TxPDO3 Trace	TraceConfiguration	Nuevo Nodo

Tabla 4.3. Numeración TxPDOs para el nuevo nodo.
La tabla muestra la numeración utilizada para el envío de los mensajes PDO mediante la API de canfestival

Índice	Tipo de PDO	Objeto que transporta	Nombre del nodo esclavo
0x1400	RxPDO1	Statusword	rightWheels
0x1401	RxPDO2 Faulhaber	FaulhaberData	rightWheels
0x1402	RxPDO3 Trace	TraceData	rightWheels
0x1403	RxPDO1	Statusword	leftWheels
0x1404	RxPDO2 Faulhaber	FaulhaberData	leftWheels
0x1405	RxPDO3 Trace	TraceData	leftWheels
0x1406	RxPDO1	Statusword	Nuevo Nodo
0x1407	RxPDO2 Faulhaber	FaulhaberData	Nuevo Nodo
0x1408	RxPDO3 Trace	TraceData	Nuevo Nodo

Tabla 4.4. Índices de los mensajes RxPDO para el nuevo nodo
La tabla muestra los índices utilizados para el envío de peticiones mediante mensajes PDO.

5. Aportaciones, conclusiones y trabajos futuros.

Este proyecto ha supuesto el diseño y desarrollo de un controlador que permite la comunicación con motores mediante el protocolo CANopen. Este controlador implementa todas las funcionalidades propias de los dispositivos de control de motores. Además, el controlador ha sido instalado en el nuevo robot móvil Mopi para el manejo de los motores de su base mediante un mando inalámbrico o joypad. Para la integración del controlador CANopen con otras aplicaciones del robot se ha utilizado el framework ROS. Por tanto, las aportaciones de este proyecto han sido:

- Desarrollo de un controlador CANopen que permite el control de motores mediante el protocolo de comunicaciones CANopen. En concreto, el dispositivo utilizado ha sido el modelo MCDC3006C del fabricante Faulhaber [12]. Para este controlador, se ha creado la librería llamada *canopen_driver*, que implementa todas las funcionalidades de los dispositivos de control y permite la creación de una red CANopen. Esta librería, *canopen_driver*, se ha diseñado mediante una arquitectura software modular permitiendo su reutilización para la implementación de las funcionalidades extras de otros dispositivos de control de motores. Por otro lado, este controlador permite la sustitución del hardware por el de otro fabricante, que también utilice CANopen, sin necesidad de reconfigurar la red CANopen. Esta sustitución puede realizarse fácilmente puesto que el protocolo CANopen especifica el funcionamiento que deben tener los diferentes tipos de dispositivos.

- Este controlador CANopen se ha sido diseñado para su integración en el framework ROS permitiendo su compatibilidad con otras aplicaciones y servicios que usan este framework. El controlador constituye un nodo ROS que se ha llamado *ROSCanfestivalNode* y se comunica con los demás nodos de la red ROS mediante una serie de topics de entrada y salida.

- Una vez comprobado el funcionamiento del controlador, éste ha sido instalado en el robot Mopi permitiendo el manejo de los motores de su base. Para establecer la operatividad del robot mediante un mando inalámbrico o joypad se ha diseñado el nodo ROS *JoyMopiController*. Este nodo envía las órdenes pertinentes a los controladores CANopen dependiendo del botón pulsado en el mando inalámbrico permitiendo la realización de diferentes movimientos.

La utilización del protocolo CANopen en el robot Mopi supone una ventaja debido a su reducción en el cableado y simplificación de las comunicaciones. Todos los dispositivos de una red CANopen son conectados a un mismo bus de comunicación mediante el cual se transmiten tramas de datos con la misma estructura, independientemente del tipo de dispositivo conectado.

Es importante destacar el framework de control ROS. Este framework se caracteriza por su modularidad, el gran número de herramientas de programación que facilita y además suministra soporte para un gran número de dispositivos. Existe una gran comunidad que está utilizando este framework de programación por lo que cada día está creciendo el número de aplicaciones implementadas. La facilidad en su estructura de comunicaciones entre las diferentes aplicaciones ha facilitado el desarrollo de este proyecto.

El principal inconveniente que se ha encontrado en el uso del protocolo de comunicaciones CANopen ha radicado en que los fabricantes no siempre cumplen o implementan completamente este protocolo. Para las pruebas de funcionamiento se han utilizado dos dispositivos de control de motores de fabricantes diferentes, en concreto el dispositivo *MCDC3006C* del fabricante *Faulhaber* y el dispositivo *ISC4805* del fabricante *Technosoft* [15]. Ninguno de los dos fabricantes cumple el protocolo en su totalidad. Esta falta en el cumplimiento ha supuesto variaciones en el software implementado para permitir el uso todas las funcionalidades de ambos dispositivos. Además, cada fabricante busca la diferenciación con sus competidores mediante la implementación de nuevas funcionalidades. Esto provoca que, además de la implementación del protocolo CANopen en sus dispositivos, desarrollen funciones específicas dentro de los márgenes permitidos por CANopen. Por tanto es recomendable diseñar el software lo más genérico posible evitando el uso de estas funcionalidades específicas que supondrían la utilización de dispositivos de un determinado fabricante.

A pesar del inconveniente anteriormente comentado, se recomienda seguir con esta línea de trabajo, iniciada en este proyecto, que combina el framework ROS junto con el protocolo de comunicaciones CANopen. Las futuras vías de trabajo que se proponen son:

- Ampliación de las funcionalidades del nodo *JoyMopiController.* Es interesante continuar desarrollando este nodo permitiendo el uso de todas las funcionalidades de los motores a través de su mando inalámbrico.

- Implementación de la capa de configuración de servicios LSS (Layer Setting Services) para CANopen. Esta capa permitirá el cambio del número de identificación de los nodos esclavos y su velocidad de transmisión de datos sin la necesidad de utilizar las aplicaciones de cada fabricante. Esto ofrecerá una mayor flexibilidad a la hora de configurar la red. Las especificaciones de dicha capa se encuentran en el documento de la CiA 305 DSP V2.2 CANopen layer setting services (LSS) and protocols [16].

- Instalación y configuración de las tarjetas de adquisición de datos Micromod Mix 3 en la red CANopen creada en este proyecto. Esto permitirá al robot Mopi, por ejemplo, la incorporación de sensores para interaccionar con su entorno. El funcionamiento de este tipo de dispositivos viene definido en el perfil de la CiA 401 V3.0.0: CANopen device profile for generic I/O modules [17].

6. Bibliografía

[1] Robot Operating System, «www.ros.org,» [En línea]. [Último acceso: 04 2011].

[2] H. Zimmermann, «OSI Reference Model. IEEE Transactions on Communications,» 1980.

[3] Canfestival, «www.canfestival.org,» [En línea].

[4] Electronic Industries Alliance , «http://www.ecaus.org,» [En línea].

[5] International Organization for Standardization, «www.iso.org,» 2003. [En línea]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=33422.

[6] International Organization for Standardization, «www.iso.org,» 2003. [En línea]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=33423.

[7] CAN in Automation, «http://www.can-cia.org/,» [En línea].

[8] CAN in Automation, «Profile DS301,» 2002.

[9] CAN in Automation, «Profile DS402».

[10] Peak System, «http://www.peak-system.com/,» 2011. [En línea].

[11] Faulhaber, «http://www.faulhaber.com/,» 2011. [En línea].

[12] Faulhaber, «Instruction manual Serie MCDC 3003/06,» 1 10 2009. [En línea]. Available: http://www.faulhaber.com/n41656/i600982.html.

[13] Lolitech, «http://www.lolitech.fr/,» [En línea].

[14] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler y A. Y. , «ROS: an open-source Robot Operating System. In Open-source software workshop,» (2009). [En línea]. Available: http://ai.stanford.edu/~mquigley/papers/icra2009-ros.pdf.

[15] Technosoft, «http://www.technosoftmotion.com/,» [En línea].

[16] CAN in Automation, «Layer Setting Services and Protocol (LSS),» 2000.

[17] CAN in Automation, «Device profile for generic I/O modules. CiA 401 V3.0.0,» 2008.

[18] F. Dupin, «CANopen Memento,» 2009.

[19] Embedded Systems Academy, «http://www.canopenmagic.com/,» [En línea].

Anexos

A. Diccionario de objetos de los nodos maestros

En la Tabla 2.3 puede observarse las diferentes agrupaciones de objetos según su funcionalidad dentro del diccionario de objetos. A continuación se muestra más detalladamente, en la tabla A.1, los diferentes objetos dentro del diccionario de objetos de los nodos maestros que permite la comunicación CANopen con los nodos esclavos.

Index	Nombre del Objeto	SubIndex	Parameter Name	Parameter Value
0x1280	SDO 1 Parameter	0x00	Número de entradas	3
		0x01	COBID Client to Server	0x660
		0x02	COBID Server to Client	0x5E0
		0x03	NodeID of the Server	0x60
0x1281	SDO 2 Parameter	0x00	Número de entradas	3
		0x01	COBID Client to Server	0x661
		0x02	COBID Server to Client	0x5E1
		0x03	NodeID of the Server	0x61
0x1400	Receive PDO 1 Parameter	0x00	Número de entradas	5
		0x01	COBID RxPDO	0x1E0
		0x02	Transmission Type	0xFD
		0x03	Inhibit Time	0x00
		0x04	Compatibility Entry	0x00
		0x05	Event Timer	0x00
0x1401	Receive PDO 2 Parameter	0x00	Número de entradas	5
		0x01	COBID RxPDO	0x2E0
		0x02	Transmission Type	0xFD
		0x03	Inhibit Time	0x00
		0x04	Compatibility Entry	0x00
		0x05	Event Timer	0x00
0x1402	Receive PDO 3 Parameter	0x00	Número de entradas	5
		0x01	COBID RxPDO	0x3E0
		0x02	Transmission Type	0xFD
		0x03	Inhibit Time	0x00
		0x04	Compatibility Entry	0x00
		0x05	Event Timer	0x00
0x1403	Receive PDO 4 Parameter	0x00	Número de entradas	5
		0x01	COBID RxPDO	0x1E1
		0x02	Transmission Type	0xFD
		0x03	Inhibit Time	0x00
		0x04	Compatibility Entry	0x00
		0x05	Event Timer	0x00
0x1404	Receive PDO 5 Parameter	0x00	Número de entradas	5
		0x01	COBID RxPDO	0x2E1
		0x02	Transmission Type	0xFD
		0x03	Inhibit Time	0x00
		0x04	Compatibility Entry	0x00

		0x05	Event Timer	0x00
0x1405	Receive PDO 6 Parameter	0x00	Número de entradas	5
		0x01	COBID RxPDO	0x3E1
		0x02	Transmission Type	0xFD
		0x03	Inhibit Time	0x00
		0x04	Compatibility Entry	0x00
		0x05	Event Timer	0x00
0x1600	Receive PDO1 Mapping	0x00	Number of Entries	1
		0x01	Statusword (0x6040)	0x00
0x1601	Receive PDO2 Mapping	0x00	Number of Entries	6
		0x01	Command (0x2306)	0x00
		0x02	LLB (0x2306)	0x00
		0x03	LHB (0x2306)	0x00
		0x04	HLB (0x2306)	0x00
		0x05	HHB (0x2306)	0x00
		0x06	Error (0x2306)	0x00
0x1602	Receive PDO3 Mapping	0x00	Number of Entries	3
		0x01	tdValueParameter1 (0x2304)	0x00
		0x02	tdValueParameter2 (0x2304)	0x00
		0x03	tdTimeCode (0x2304)	0x00
0x1603	Receive PDO4 Mapping	0x00	Number of Entries	1
		0x01	Statusword(0x6040)	
0x1604	Receive PDO5 Mapping	0x00	Number of Entries	6
		0x01	Command (0x2306)	
		0x02	LLB (0x2306)	
		0x03	LHB (0x2306)	
		0x04	HLB (0x2306)	
		0x05	HHB (0x2306)	
		0x06	Error (0x2306)	
0x1605	Receive PDO6 Mapping	0x00	Number of Entries	3
		0x01	tdValueParameter1 (0x2304)	
		0x02	tdValueParameter2 (0x2304)	
		0x03	tdTimeCode (0x2304)	
0x1800	Transmit PDO1 Parameter	0x00	Número de entradas	5
		0x01	COBID RxPDO	0x260
		0x02	Transmission Type	0xFF
		0x03	Inhibit Time	0x00
		0x04	Compatibility Entry	0x00
		0x05	Event Timer	0x00
0x1801	Transmit PDO2	0x00	Número de entradas	5

	Parameter	0x01	COBID RxPDO	0x360
		0x02	Transmission Type	0xFF
		0x03	Inhibit Time	0x00
		0x04	Compatibility Entry	0x00
		0x05	Event Timer	0x00
0x1802	Transmit PDO3 Parameter	0x00	Número de entradas	5
		0x01	COBID RxPDO	0x460
		0x02	Transmission Type	0xFF
		0x03	Inhibit Time	0x00
		0x04	Compatibility Entry	0x00
		0x05	Event Timer	0x00
0x1803	Transmit PDO4 Parameter	0x00	Número de entradas	5
		0x01	COBID RxPDO	0x261
		0x02	Transmission Type	0xFF
		0x03	Inhibit Time	0x00
		0x04	Compatibility Entry	0x00
		0x05	Event Timer	0x00
0x1804	Transmit PDO5 Parameter	0x00	Número de entradas	5
		0x01	COBID RxPDO	0x361
		0x02	Transmission Type	0xFF
		0x03	Inhibit Time	0x00
		0x04	Compatibility Entry	0x00
		0x05	Event Timer	0x00
0x1805	Transmit PDO6 Parameter	0x00	Número de entradas	5
		0x01	COBID RxPDO	0x461
		0x02	Transmission Type	0xFF
		0x03	Inhibit Time	0x00
		0x04	Compatibility Entry	0x00
		0x05	Event Timer	0x00
0x1A00	Transmit PDO1 Mapping	0x00	Number of Entries	1
		0x01	Controlword(0x6040)	0x00
0x1A01	Transmit PDO2 Mapping	0x00	Number of Entries	5
		0x01	Command (0x2305)	0x00
		0x02	LLB (0x2305)	0x00
		0x03	LHB (0x2305)	0x00
		0x04	HLB (0x2305)	0x00
		0x05	HHB (0x2305)	0x00
0x1A02	Transmit PDO3 Mapping	0x00	Number of Entries	5
		0x01	tcMode1 (0x2303)	0x00
		0x02	tcMode1 (0x2303)	0x00
		0x03	tcTimeCode (0x2303)	0x00
		0x04	tcPackets (0x2303)	0x00
		0x05	tcPeriod (0x2303)	0x00
0x1A03	Transmit PDO4 Mapping	0x00	Number of Entries	1
		0x01	Controlword(0x6040)	0x00
0x1A04	Transmit PDO5	0x00	Number of Entries	5

	Mapping	0x01	Command (0x2305)	0x00
		0x02	LLB (0x2305)	0x00
		0x03	LHB (0x2305)	0x00
		0x04	HLB (0x2305)	0x00
		0x05	HHB (0x2305)	0x00
0x1A05	Transmit PDO6 Mapping	0x00	Number of Entries	5
		0x01	tcMode1 (0x2303)	0x00
		0x02	tcMode1 (0x2303)	0x00
		0x03	tcTimeCode (0x2303)	0x00
		0x04	tcPackets (0x2303)	0x00
		0x05	tcPeriod (0x2303)	0x00
0x2303	traceConfiguration	0x00	Number of Entries	5
		0x01	tcMode1	0x00
		0x02	tcMode1	0x00
		0x03	tcTimeCode	0x01
		0x04	tcPackets	0x01
		0x05	tcPeriod	0x01
0x2304	traceData	0x00	Number of Entries	3
		0x01	tdValueParameter1	0x00
		0x02	tdValueParameter2	0x00
		0x03	tdTimeCode	0x00
0x2305	faulhaberCommand	0x00	Number of Entries	5
		0x01	Command	0x00
		0x02	LLB	0x00
		0x03	LHB	0x00
		0x04	HLB	0x00
		0x05	HHB	0x00
0x2306	faulhaberData	0x00	Number of Entries	6
		0x01	Command	0x00
		0x02	LLB	0x00
		0x03	LHB	0x00
		0x04	HLB	0x00
		0x05	HHB	0x00
		0x06	Error	0x00
0x6040	Controlword	0x00	Controlword	0x00
0x6041	Statusword	0x00	Statusword	0x00

Tabla A.1 Diccionario de objetos del maestro
Se muestran los objetos del diccionario de objetos de los maestros que definen sus comunicaciones.

B. Implementación del diccionario de objetos del nodo maestro

Canfestival incorpora una aplicación gráfica que permite crear con facilidad un diccionario de objetos y generar posteriormente todo el código necesario. Esta aplicación se encuentra *en ${CANFESTIVAL_INSTALL_DIR}/objdictgen* y para ejecutarla se utiliza el siguiente comando:

$ python objdictedit.py

Se abrirá una ventana como la que muestra la ilustración B.1:

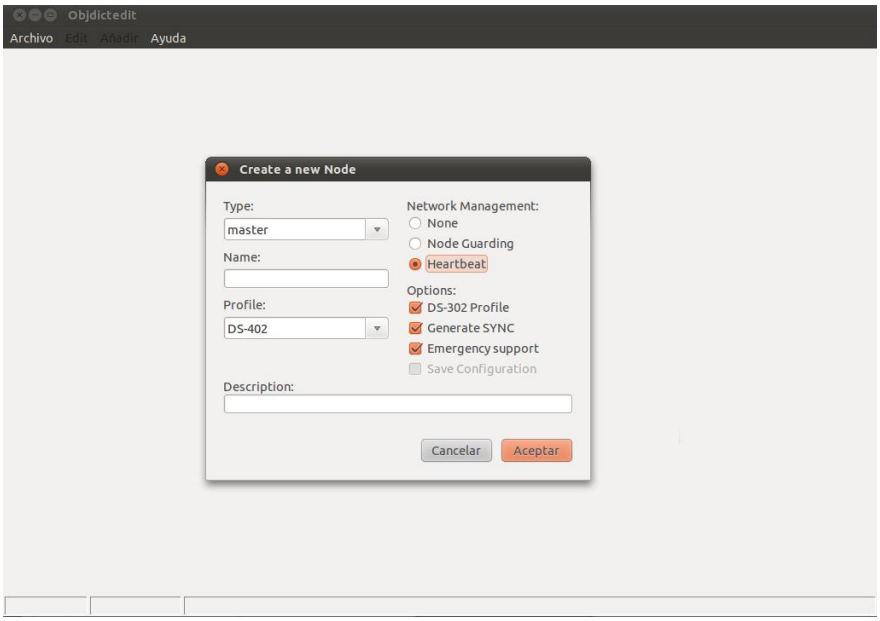


Ilustración B.1 Aplicación objdictgen. Pantalla de inicio

En esta pantalla de inicio se determina el tipo de nodo al cual va asignado el diccionario que vamos a crear, en el caso de este proyecto se tratan de nodos maestros. Se determina el nombre del nodo y el perfil que va a definir el diccionario de objetos, en este caso al tratarse de dispositivos controladores la CiA (CAN in Automation) recoge sus especificaciones en el archivo DS-402 [9].

A la derecha puede escogerse el tipo control de la red que puede ser: **Node Guarding y Heartbeat.** Ambos son protocolos de comunicación que permiten al maestro saber los nodos esclavos están en funcionamiento.

También es posible elegir las siguientes funciones:

Ds-302 profile: Los mecanismos adicionales de comunicación especificados en el perfil DS302 son utilizados para dispositivos inteligentes como PLCs, MMIs y herramientas CANopen tools.

Generate SYNC: Utilizado para el envío síncrono de eventos PDO.
Emergency support: Ver apartado 2.4.7.

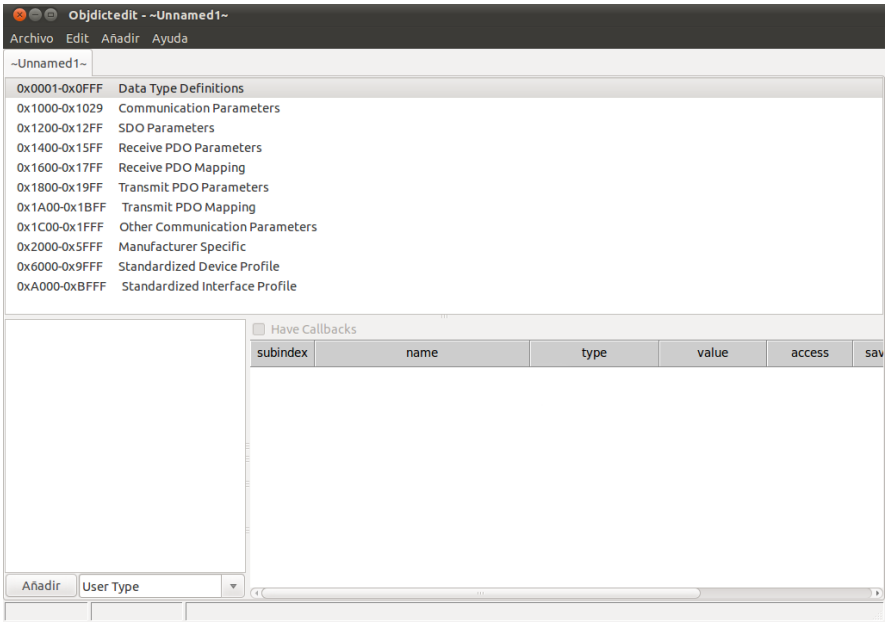


Ilustración B.2 Aplicación objdictgen. Diccionario de objetos

En la Ilustración B.2 pueden observarse cada una de las partes del diccionario de objetos:

- **Data Type Definitions:** Se encuentran los objetos que definen los diferentes tipos de datos.
- **Communication Parameters:** Engloba los objetos relativos a la comunicación.
- **SDO Parameters:** Define los parámetros de comunicación de los mensajes SDO.
- **Receive PDO Parameters:** Define los parámetros de comunicación de los mensajes de recepción PDO.
- **Receive PDO Mapping:** Determina los objetos que se reciben mediante los mensajes RxPDO.
- **Transmit PDO Parameters:** Define los parámetros de comunicación de los mensajes de recepción PDO.
- **Transmit PDO Mapping:**
- **Other Communication Parameters:**
- **Manufacturer Specific: Objetos específicos del fabricante.**
- **Standardized Device Profile:** Objetos definidos por los perfiles de la CiA para determinados tipos de dispositivos.
- **Standardized Interface Profile:** Objetos definidos por los perfiles de la CiA para definir la interfaz.

- **SDO Parameters:**

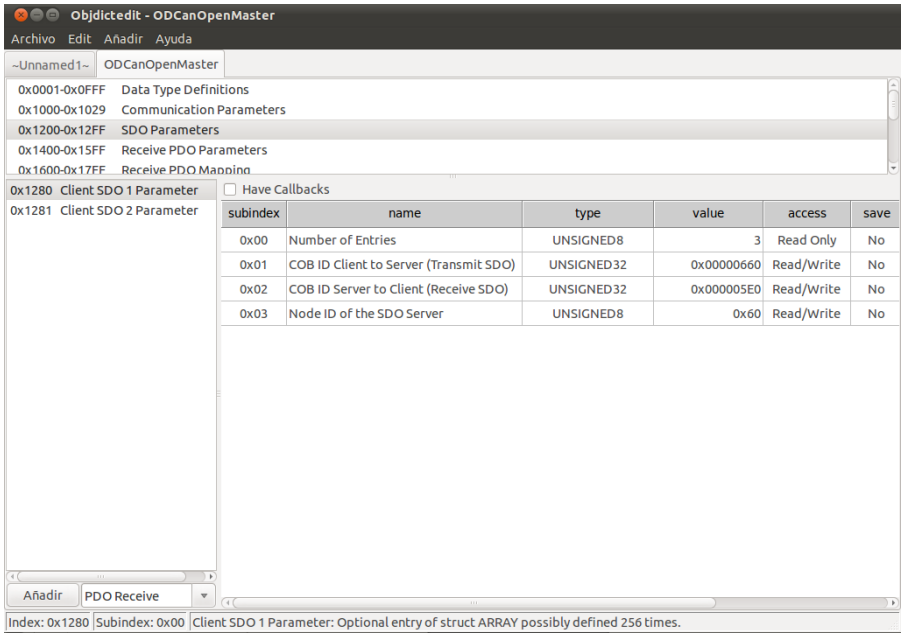


Ilustración B.3 Aplicación objdictgen. SDO Parameters.

La Ilustración B.3 muestra los parámetros relacionados con los objetos SDO, en concreto muestra los identificadores de los mensajes SDO para comunicarse con el nodo identificación 0x60. Debe existir un objeto por cada nodo esclavo, por tanto para el proyecto se han creado dos objetos SDO. Para crearlos se pulsa en el botón añadir que aparece abajo a la izquierda y se escoge la opción Client SDO ya que se está creando el diccionario del nodo maestro el cual dentro de protocolo SDO constituye el cliente y el nodo esclavo el servidor.

Una vez creados los objetos hay que definir sus parámetros:

COBID Client to Server (Transmit SDO): debe coincidir con el Receive SDO del nodo esclavo, el cual lleva la asignación por defecto 0x600 + NodeID.

COBID Server to Client (Receive SDO): debe coincidir con el Transmit SDO del nodo esclavo, el cual lleva la asignación por defecto 0x580 + NodeID.

NodeID of the SDO Server: Número de identificación, dentro de la red CANopen, del nodo esclavo (Servidor).

- **Receive PDO Parameters.**

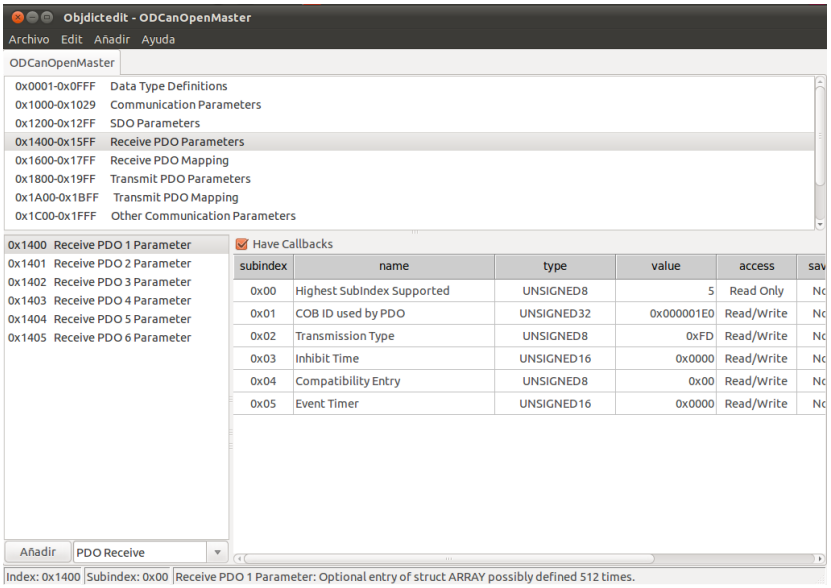


Ilustración B.4 Aplicación objdictgen. Receive PDO Parameters.

La Ilustración B.4 muestra los objetos PDO de recepción los cuales establecen los parámetros de comunicación de los mensajes RxPDO. Los drivers mcdc3006 utilizados en el proyecto poseen tres RxPDO cada uno por lo que se crean seis objetos RxPDO para recibir los mensajes de los dos nodos utilizados. Los COBID utilizados para los mensajes RxPDO del maestro en este proyecto se muestran en la Ilustración 4.12.

- **Receive PDO Mapping.**

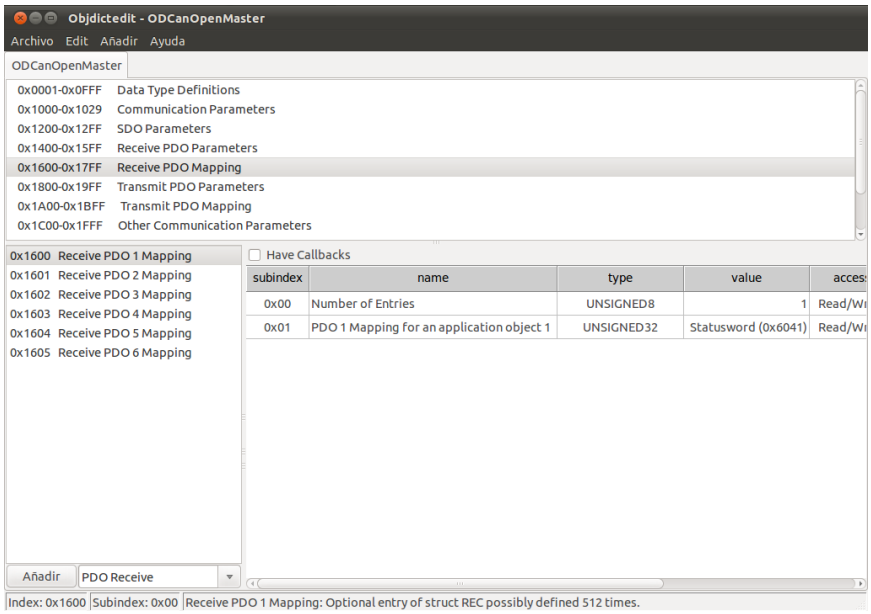


Ilustración B.5 Aplicación objdictgen. Receive PDO Mapping.

Mediante el objeto Receive PDO Mapping se establecen que objetos serán recibidos mediante los mensajes RxPDO. Como muestra la Ilustración B.5 mediante el RxPDO1 se recibirá el objeto Statusword del driver mcdc3006c indicando el estado del driver.

- **Transmit PDO Parameters.**

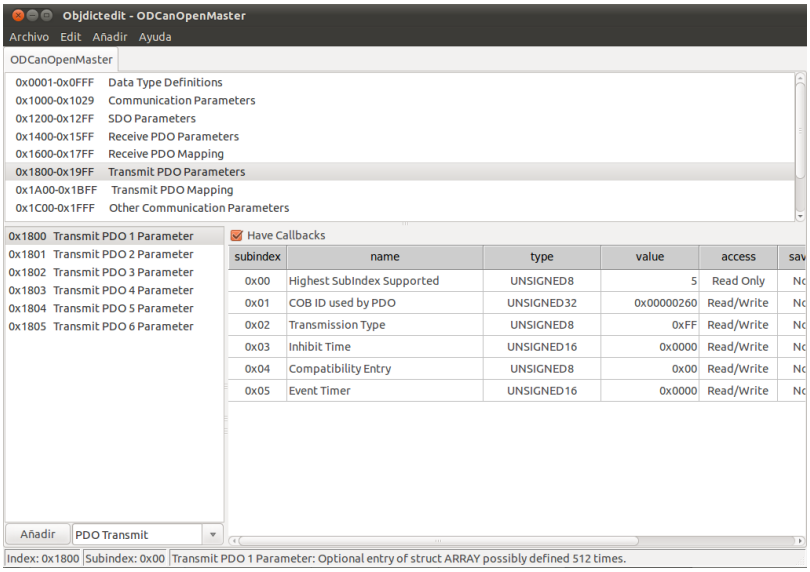


Ilustración B.6 Aplicación objdictgen. Transmit PDO Parameters.

En este caso la Ilustración B.6 muestra los parámetros de comunicación de los TxPDO que deben configurarse para garantizar la comunicación de los esclavos con el maestro. Los COBID utilizados para los mensajes TxPDO del maestro en este proyecto se muestran en la Ilustración 4.12.

- **Transmit PDO Mapping**

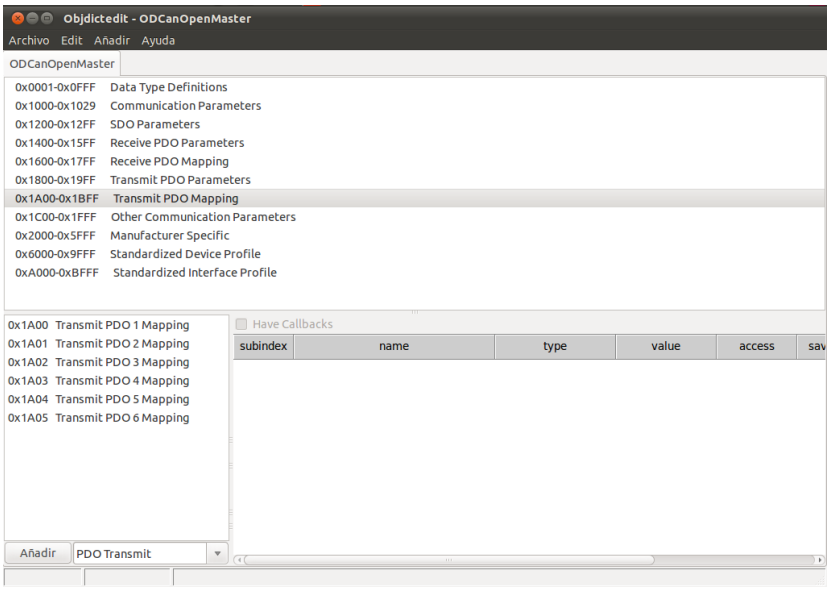


Ilustración B.7 Aplicación objdictgen. Transmit PDO Mapping.

En la Ilustración B.7 deben configurarse los objetos que van a ser transmitidos por el nodo. En este proyecto los objetos que ha transmitido el maestro han sido el objeto *Controlword* (0x6040) mediante el TxPDO1 y TxPDO2, el objeto *FaulhaberCommand* (0x2303) mediante los mensajes TxPDO2 y TxPDO3 y el objeto *TraceCommand* (0x2305) mediante los mensajes TxPDO4 y TxPDO5.

- **Manufacturer Specific**

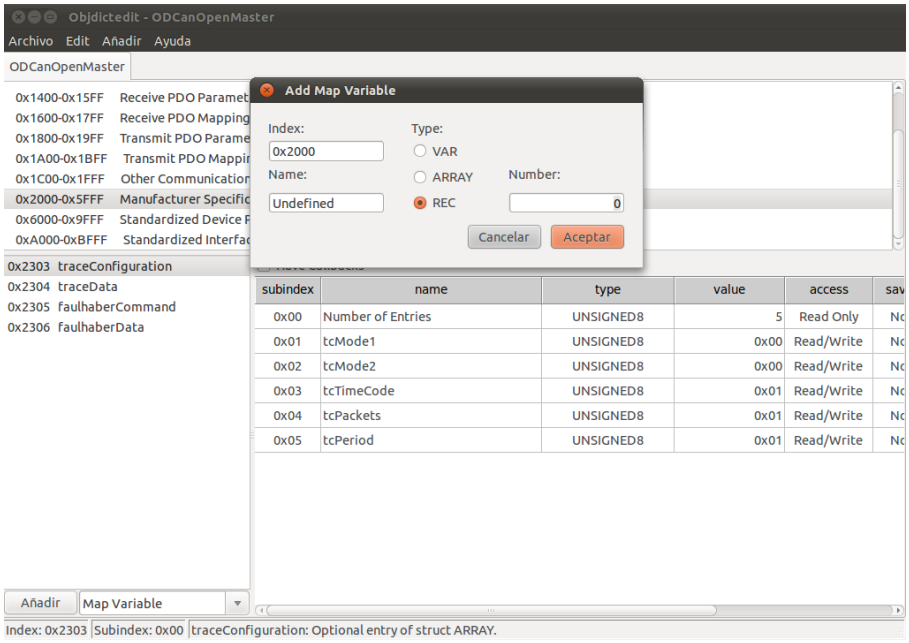


Ilustración B.8 Aplicación objdictgen.Manufacturer Specific.

En la Ilustración B.8 pueden observarse los objetos creados propios del fabricante Faulhaber. Para crear dichos objetos debe pulsarse el botón añadir y aparecerá la ventana Add Map Variable en la que se escogerá la entrada y el nombre de dicho objeto. Si el objeto tiene un único subíndice el tipo de objeto que hay que escoger es Var. En caso de tratarse de un objeto con más de un subíndice el tipo será Array o Rec y en la casilla Number deberá introducirse el número de subíndices de dicho objeto. La única diferencia entre el tipo Array y Rec es que este último permite nombrar cada uno de los subíndices mientras el primero fija el nombre automáticamente.

- **Standardized Device Profile**

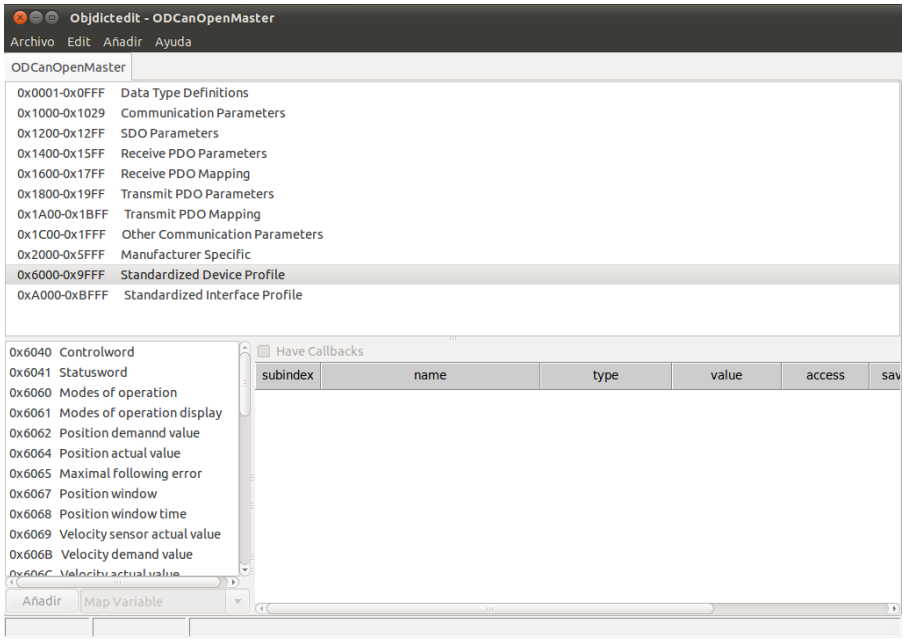


Ilustración B.9 Aplicación objdictgen. Standardized Device Profile

La Ilustración B.9 muestra la sección del diccionario en la que aparecerán todos los objetos definidos en el perfil de la CíA escogido anteriormente. En este caso se escogido el perfil DS-402 por lo que los objetos que aparecen son los utilizados por dispositivos de control de motores.

Una vez terminada edición del diccionario de objetos es necesario crear el archivo en código C que será adjuntado a *ROSCanfestivalNode*, para ello en el menú Archivo se selecciona Build Dicctionary con lo que se creará tanto el archivo diccObj.h como diccObj.c.

C. Descripción de los Mensajes ROS

Para el transporte de los datos se han creado los siguientes tipos de mensajes que definen los tipos de estos datos, según su funcionalidad se han creado tres tipos de mensajes que son:

- ***motorConfiguration.msg:*** Define los tipos de las variables relacionadas con la configuración:

```
Header header
float64 maxPosition          #Máxima posición del motor en rad.
float64 minPosition          #Mínima posición del motor en rad.
float64 maxVelocity          #Máxima velocidad del motor en rad/s.
float64 maxAcceleration   #Máxima aceleración del motor en rad/s^2.
float64 maxDecceleration     #Máxima deceleración del motor en rad/s^2.
string driverName         #Nombre del driver.
int8   driverID              #Número de identificador CANopen del motor.
string driverType            #Tipo de driver.
string baud               #Velocidad de transmisión de datos
```

- ***motorSensor.msg:*** Define los tipos de las variables relacionadas con el sensor:

```
Header header
float64 position             #valor de la posición del motor en rad.
float64 velocity             #valor de la velocidad del motor en ra/sd.
int32 current                #valor de la corriente del motor en mA.
```

- ***motorStatus.msg:*** Define los tipos de las variables relacionadas con el estado del motor:

```
Header header
bool enabled                  #Habilita e inhabilita el motor.
bool calibrated               #Habilita la calibración del motor.
bool overTemperatureWarning #Máxima temperatura del driver en °C.
bool overVoltageWarning   #Máximo voltaje del driver en V.
bool currentLimitated         #Máxima corriente del motor en mA
bool limitSensorReached   #Indica si se ha alcanzado el límite de
                          #corriente
```

- **move.msg:** Define los tipos de las variables relacionadas con el movimiento del motor, es un mensaje utilizado para el envío de órdenes de movimiento al motor, contiene:

```
Header header
```

```
uint8 movement_type     # Indica el movimiento que debe realizar el motor.
                        # Según el valor de esta variable los tipos de
                        # movimientos son:

                        # 1: Movimiento por velocidad.
                        # 2: Movimiento por velocidad a una posición
                        # determinada.
                        # 3: Movimiento a una posición absoluta.
                        # 4: Movimiento por posición relative

float64 velocity        # Valor de velocidad en    rad/s
float64 position        # Valor de posición en rad
```

D. Archivo de configuración Xml

A continuación se muestra un ejemplo de un archivo de configuración, en concreto se trata del archivo que configura el driver con ID=0x97 que controla las ruedas del lado izquierdo.

```
<?xml version="1.0"?>
        <driver>
        <driverName>leftWheels</driverName>
        <driverID>97</driverID>
        <driverBaud>500K</driverBaud>
        <driverType>CANOpen</driverType>
        <mcdc3006c>

<!-- Parmameters of the class mcdc3006_canopen_configuration -->
          <maxPos>15000</maxPos>
          <minPos>-15000</minPos>
          <maxVel>2500</maxVel>
          <calibrationSensorPosition>512</calibrationSensorPosition>
          <minVolt>7</minVolt>
          <maxVolt>12.1</maxVolt>
          <minTemp>-5</minTemp>
          <maxTemp>85</maxTemp>
          <continousCurrentLimit>200</continousCurrentLimit> <!-- mA -->
          <peakCurrentLimit>250</peakCurrentLimit> <!-- mA -->
          <encPulsesPerRev>2048</encPulsesPerRev>
          <maxAcceleration>1000</maxAcceleration>
          <maxDecceleration>1000</maxDecceleration>
          <polarity>positive</polarity>
          <positionEncoderResolution>2048</positionEncoderResolution>
          <quickStopDeceleration>30000</quickStopDeceleration>
          <dividendPositionFactor>1</dividendPositionFactor>
          <divisorPositionFactor>1</divisorPositionFactor>
          <dividendVelocityFactor>1</dividendVelocityFactor>
          <divisorVelocityFactor>1</divisorVelocityFactor>
          <dividendAccelerationFactor>1</dividendAccelerationFactor>
          <divisorAccelerationFactor>1</divisorAccelerationFactor>

<!-- Parmameters of the class mcdc3006_canopen_params -->
<!-- These Parameters are used for sending PDO and request PDO -->
          <numberPDO1>3</numPD1>
          <indexRPDO1>5123</indexRPDO1> <!-- RPDO1 0x1403= 5120d -->
          <numberPDO2>4</numPD2>
          <indexRPDO2>5124</indexRPDO2> <!-- RPDO1 0x1404 = 5121d -->
          <numberPDO3>5</numberPDO3>
          <indexRPDO3>5125</indexRPDO3> <!-- RPDO1 0x1405 = 5123d -->
          </mcdc3006c>
          </driver>
```

Pueden observarse tres grupos de datos de configuración. Los primeros son utilizados por el controlador para el establecimiento de las comunicaciones tanto en su nivel ROS como en el nivel CANopen. El segundo grupo de parámetros permitirá establecer la configuración de funcionamiento del motor, para ello son enviados a través de la red CANopen al dispositivo de control de motores. Todos los nodos maestros de la red CANopen comparten el mismo diccionario de objeto en el que se definen todos los identificadores de mensajes o COBIDs que permiten la comunicación CANopen con los esclavos. Para enviar un evento PDO es necesario indicar la posición del mensaje PDO que se desea enviar y en el caso de una petición se utiliza el índice de su entrada. Por tanto Los parámetros numberPDO e indexRPDO permiten a cada nodo maestro enviar el PDO del diccionario de objetos que comunica con el nodo esclavo asignado y la entrada del diccionario de objeto donde se define que objeto se recibirá mediante el mensaje RxPDO.

E. Transferencia expedita

Esta transferencia mediante mensajes SDO se ha utilizado tanto para el trasporte tanto de órdenes como de datos de funcionamiento de los controladores CANopen a los dispositivos físicos de control MCDC3006 dentro de la red CANopen y viceversa.

Se trata de un tipo de transferencia de segmentos en la que se envían menos de cuatro bytes de datos. Por esta razón es posible realizar la transferencia de datos en una sola fase denominada *Initiate SDO Download.* Cuando la comunicación, mediante mensajes SDO, se utiliza para que el cliente escriba en el diccionario de objetos del servidor se denomina *Download*. En caso de lectura, *Upload*.

Initiate SDO Download

El cliente realiza una petición al servidor para que se prepare para la escritura de datos en su diccionario de objetos. El servicio es confirmado. En caso de fallo se ejecuta el servicio Abort SDO transfer. Si no existen errores durante el proceso de escritura este servicio finaliza la transferencia.

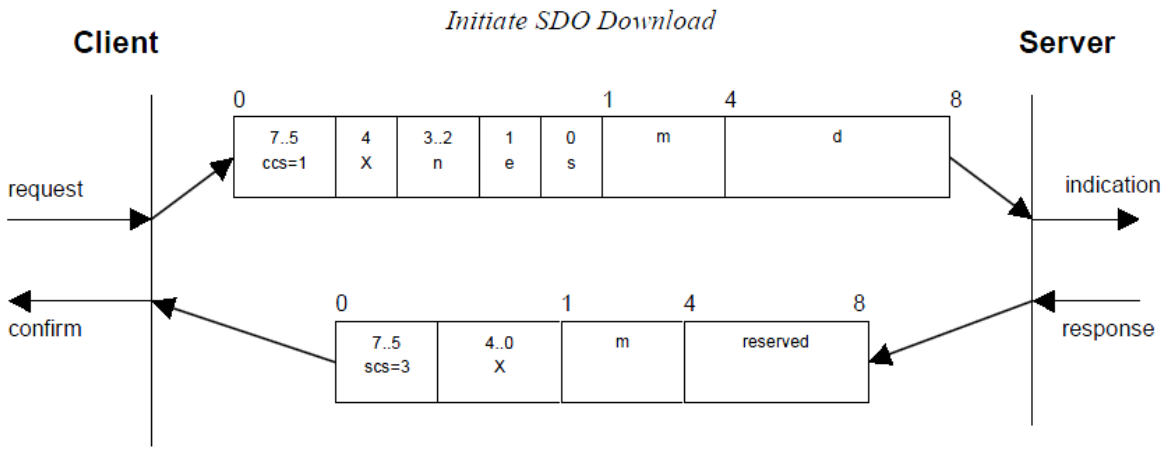


Ilustración E.1. Protocolo Initiate SDO Download.
Ilustración obtenida de CiA **[7]**.

ccs: Comando específico del cliente (client command specifier)
ccs = 1: initiate download request.
scs: Comando específico del servidor (server command specifier)
ccs = 3: initiate download response.
n: Indica el número de bytes en el campo de datos que no contienen datos, utilizado solo cuando e=1 y s=1.
e:Tipo de transferencia.
e=0: Transferencia por segmentos.
e=1: Transferencia expedita.

s (size indicator): Indicador de Tamaño.

0: Tamaño de los datos no indicado.

1: Se indica el tamaño de los datos.

m (multiplexor): Representa el índice y subíndice de la entrada del diccionario de objetos del servidor donde serán escritos los datos.

d (data): Datos

e=0 y s=0: Campo de datos reservado.

e=0 y s=1: el campo d contiene el número de bytes de datos que serán escritos en el diccionario.

e=1 y s=0: el campo d contiene un número no especificado de bytes.

e=1 y s=1: Contiene los datos que serán escritos en el diccionario. (Transferencia expedita).

X: Reservado, siempre 0.

reserved: Reservado.

A continuación se presenta un esquema de las tramas de petición de escritura mediante mensajes SDO (SDO Download) para una transferencia expedita. La Ilustración E. muestra las tramas enviadas por el cliente según el número de bytes que se deseen escribir.

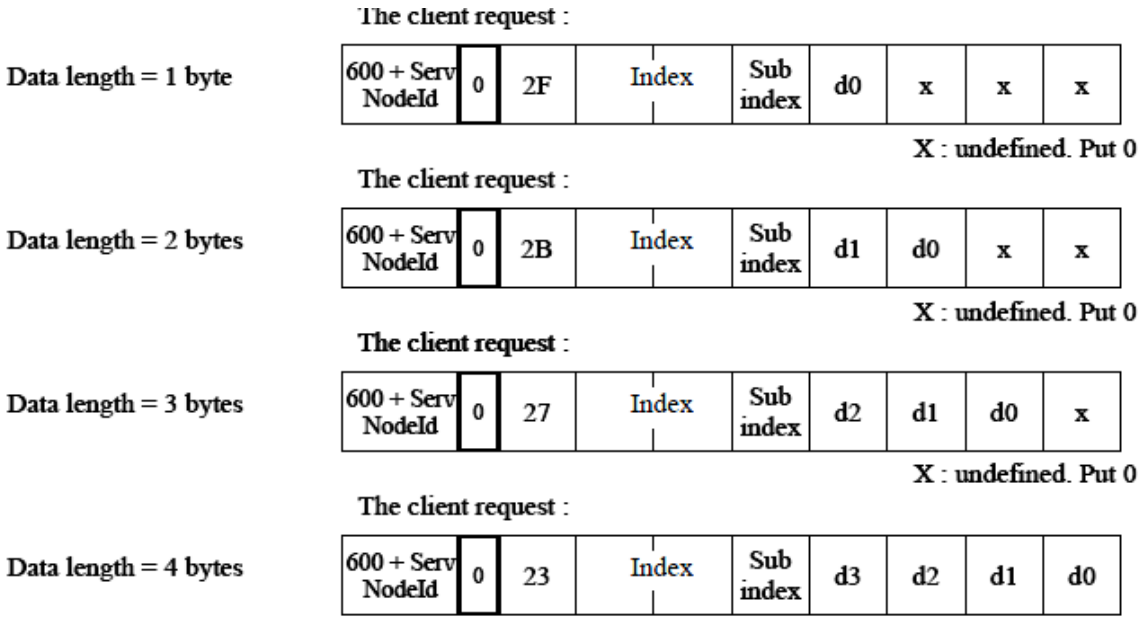


Ilustración E.2. Tramas de petición SDO Download. Transferencia expedita.
Ilustración obtenida de **[18]**

La trama de respuesta a escritura por parte del servidor para una transferencia expedita que se ha realizado con éxito se representa en la Ilustración E..

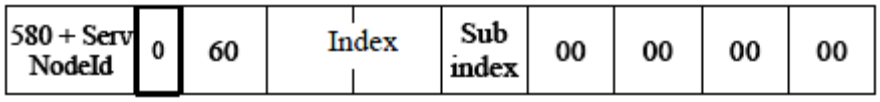


Ilustración E.3. Trama respuesta SDO Download. Transferencia expedita correcta.

O en caso de existir algún fallo el nodo responderá indicando el error, ver Ilustración E. :

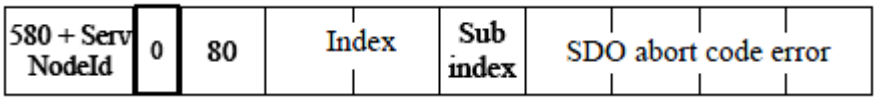


Ilustración E.4. Trama respuesta SDO Download. Transferencia expedita errónea.

Por ejemplo si se desea escribir el byte 0xFD en la entrada del diccionario de objetos cuyo índice es 0x1400 y subíndice 0x02 del nodo con número de identificación ID=0x05 se enviará la trama:

COBID	Código trama escritura	Índice	Subíndice	Dato
605	2F	00 14	02	FD 00 00 00

Si todo se realiza correctamente el nodo 0x05 enviará la siguiente trama de respuesta:

COBID	Código respuesta trama Lectura	Índice	Subíndice	Dato
585	60	00 14	02	00 00 00 00

Initiate SDO Upload

El cliente solicita al servidor que se prepare para enviarle los datos solicitados de su diccionario de objetos en el byte multiplexor. Este servicio dispone de confirmación. La trama de respuesta por parte del servidor indica si la transferencia ha sido un éxito y en caso contrario se ejecuta el servicio “Abort Transfer”.

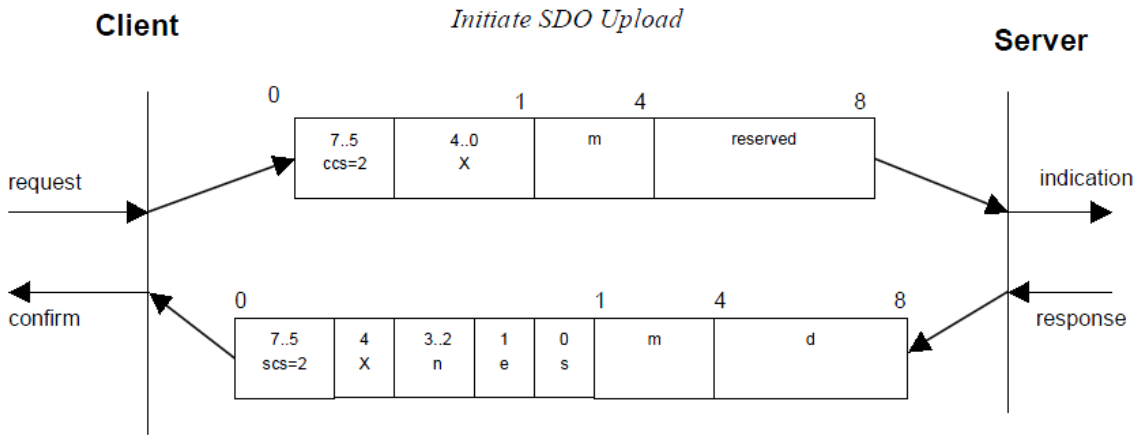


Ilustración E.5. Protocolo Initiate SDO Upload.
Ilustración obtenida de la CiA **[8]**.

ccs: Comando específico del cliente (client command specifier)
ccs = 2: initiate upload request.
scs: Comando específico del servidor (server command specifier)

ccs = 2: initiate upload response.

n: Indica el número de bytes en el campo de datos que no contienen datos, utilizado solo cuando e=1 y s=1.

e:Tipo de transferencia

e=0: Transferencia por segmentos.

E=1: Transferencia expedita

s (size indicator): Indicador de Tamaño.

0: Tamaño de los datos no indicado.

1: Se indica el tamaño de los datos.

m (multiplexor): Representa el índice y subíndice de la entrada del diccionario de objetos del servidor donde serán escritos los datos.

d (data): Datos

e=0 y s=0: campo de datos reservado.

e=0 y s=1: el campo d contiene el número de bytes de datos que serán leídos en el diccionario.

e=1 y s=0: el campo d contiene un número no especificado de bytes para ser leidos.

e=1 y s=1: contiene los datos que serán leídos en el diccionario. (Transferencia expedita)

X: Reservado, siempre 0

reserved: Reservado, siempre 0

Por tanto las tramas para el proceso de lectura de las entradas del diccionario de objetos de un nodo mediante mensajes SDO para una transferencia expedita tendrán una estructura como la que se muestra a continuación:

El cliente comenzara el proceso con el envío de la trama de petición como muestra la ilustración E.6.

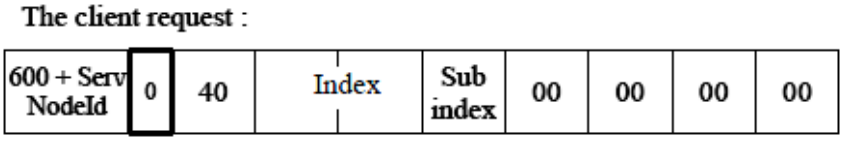


Ilustración E.6.. Trama de petición SDO Upload. Transferencia expedita. [18]

El servidor contestará con los datos pedidos en caso de éxito, ver ilustración E.7.

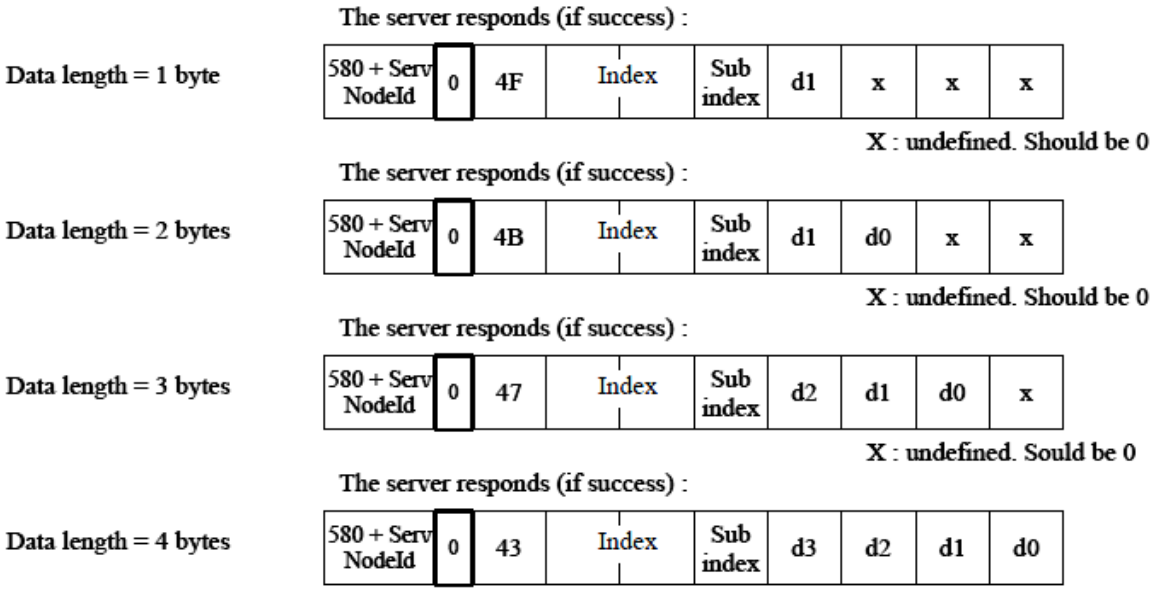


Ilustración E.7. Trama respuesta SDO Upload. Transferencia expedita correcta.

O en caso de existir algún error durante la transferencia enviará en la trama de respuesta el código con el error producido como muestra la ilustración E.8.

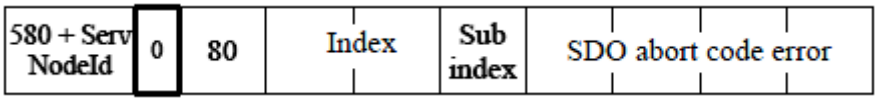


Ilustración E.8. Trama respuesta SDO Upload. Transferencia expedita errónea.

Por ejemplo si se desea leer los cuatro bytes de datos que se encuentran en la entrada del diccionario de objetos cuyo índice es 0x1603 y subíndice 0x01 del nodo con número de identificación ID=0x05 cuyo valor es 0x60120208 se enviará la trama:

COBID	Código trama Lectura	Índice	Subíndice	Dato
605	40	03 16	01	00 00 00 00

Si todo se realiza correctamente el nodo 0x05 enviará la siguiente trama de respuesta transportando en primer lugar el bit menos significativo o LSB.

COBID	Código respuesta Lectura	Índice	Subíndice	Dato
585	43	03 16	01	08 02 12 60

Abort SDO Transfer

Existe la posibilidad que al acceder a las entradas del diccionario de objetos del servidor, se produzca un error. Este protocolo es usado en esos casos para notificar tanto a clientes como a servidores. El formato de los mensajes de este protocolo es el siguiente:

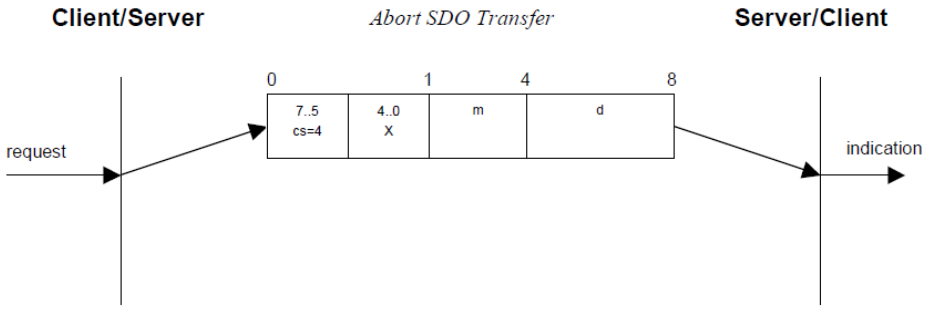


Ilustración E.9. Protocolo Abort SDO Transfer.
Muestra los diferentes elementos de una trama que indica error durante la transferencia de mensajes SDO. Ilustración obtenida de CiA **[8]**.

cs: command specifier

4: abort transfer request

X: No se usa. Su valor es cero.

m: multiplexor. Representa el índice y subíndice de la entrada del diccionario de objetos.

d: Contiene 4 bytes con el código que identifica el error.

Los diferentes códigos de error con los que se indica el fallo de la comunicación pueden observarse en la tabla E.1.

Abort code	Description
0503 0000h	Toggle bit not alternated.
0504 0000h	SDO protocol timed out.
0504 0001h	Client/server command specifier not valid or unknown.
0504 0002h	Invalid block size (block mode only).
0504 0003h	Invalid sequence number (block mode only).
0504 0004h	CRC error (block mode only).
0504 0005h	Out of memory.
0601 0000h	Unsupported access to an object.
0601 0001h	Attempt to read a write only object.
0601 0002h	Attempt to write a read only object.
0602 0000h	Object does not exist in the object dictionary.
0604 0041h	Object cannot be mapped to the PDO.
0604 0042h	The number and length of the objects to be mapped would exceed PDO length.
0604 0043h	General parameter incompatibility reason.
0604 0047h	General internal incompatibility in the device.
0606 0000h	Access failed due to an hardware error.
0607 0010h	Data type does not match, length of service parameter does not match
0607 0012h	Data type does not match, length of service parameter too high
0607 0013h	Data type does not match, length of service parameter too low
0609 0011h	Sub-index does not exist.
0609 0030h	Value range of parameter exceeded (only for write access).
0609 0031h	Value of parameter written too high.
0609 0032h	Value of parameter written too low.
0609 0036h	Maximum value is less than minimum value.
0800 0000h	general error
0800 0020h	Data cannot be transferred or stored to the application.
0800 0021h	Data cannot be transferred or stored to the application because of local control.
0800 0022h	Data cannot be transferred or stored to the application because of the present device state.
0800 0023h	Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of an file error).

Tabla E.1. Códigos de error para SDO Abort Domain Transfer. [8]

F. Montaje de verificación del funcionamiento

Para la comprobación inicial de funcionamiento se ha utilizado el montaje de la ilustración F.1. En este montaje puede observarse un ordenador donde se ejecutan los controladores CANopen junto con le red ROS de prueba a través de la cual se envía las órdenes y se observan los datos recogidos de los motores. Se ha utilizado un adaptador PCAN-USB para permitir la conexión del ordenador con el bus CAN y así poder enviar las diferentes tramas.

También se ha conectado otro ordenador al bus CAN, mediante otro adaptador PCAN-USB, en el que se está ejecutando la aplicación CANopen Magic [19]. Con esta aplicación es posible monitorizar el bus CAN permitiendo ver las tramas CAN que están siendo enviadas así como sus correspondientes tramas a nivel CANopen. Resulta de gran utilidad para la depuración de errores.

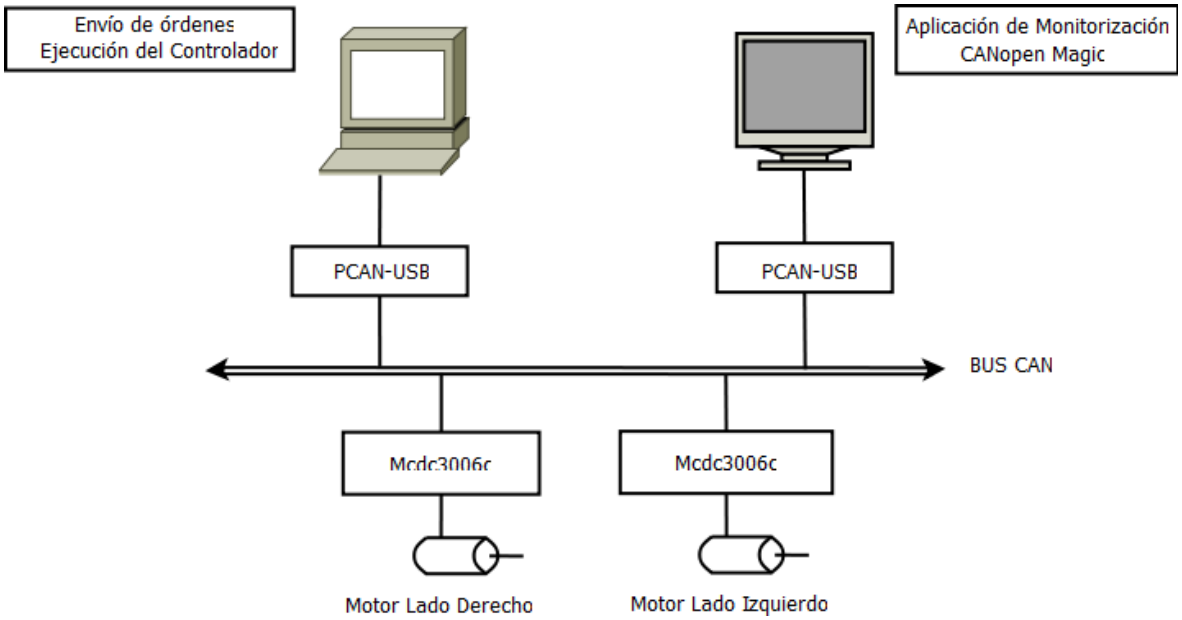


Ilustración F.1. Montaje de prueba de funcionamiento de la arquitectura software.
Montaje realizado para las pruebas de funcionamiento de los controladores CANopen y su arquitectura software.

G. Asignación del identificador al dispositivo MCDC3006C

Este capítulo explica como configurar el identificador del dispositivo MCDC3006C utilizado en las comunicaciones de la red CANopen. Este identificador puede modificarse, mediante la herramienta software suministrada por el fabricante Faulhaber [11] denominada *Motion Manager 4,* siguiendo los siguientes pasos:

1. Conexión del dispositivo MCDC3006C al bus CAN. Esta conexión se realiza mediante un conector DB-9.

2. Instalación de la herramienta software *Motion Manager* en un ordenador con sistema operativo Windows. El fabricante faulhaber solo suministra esta herramienta para sistemas operativos Windows.

3. Es necesario conectar el ordenador al bus CAN para ello se utiliza el convertidor PCAN-USB del fabricante peak-system [10]. Debe instalarse el controlador facilitado por el fabricante para el uso del convertidor. Una vez que se haya instalado todo correctamente se iluminará el led del convertido PCAN-USB.

4. Realizadas las conexiones anteriormente descrites debe iniciarse la herramienta *Motion Manage.* Para configurar el identificador del dispositivo MCDC3006C debe irse al menú CAN->LSS (DSP305) y aparecerá la ventana que se muestra en la ilustración G.2. En la ventana que aparece con el título *LSS Mode* debe seleccionarse la casilla *Globally configure individual node.* Una vez seleccionada se pulsa el botón *Ok.*

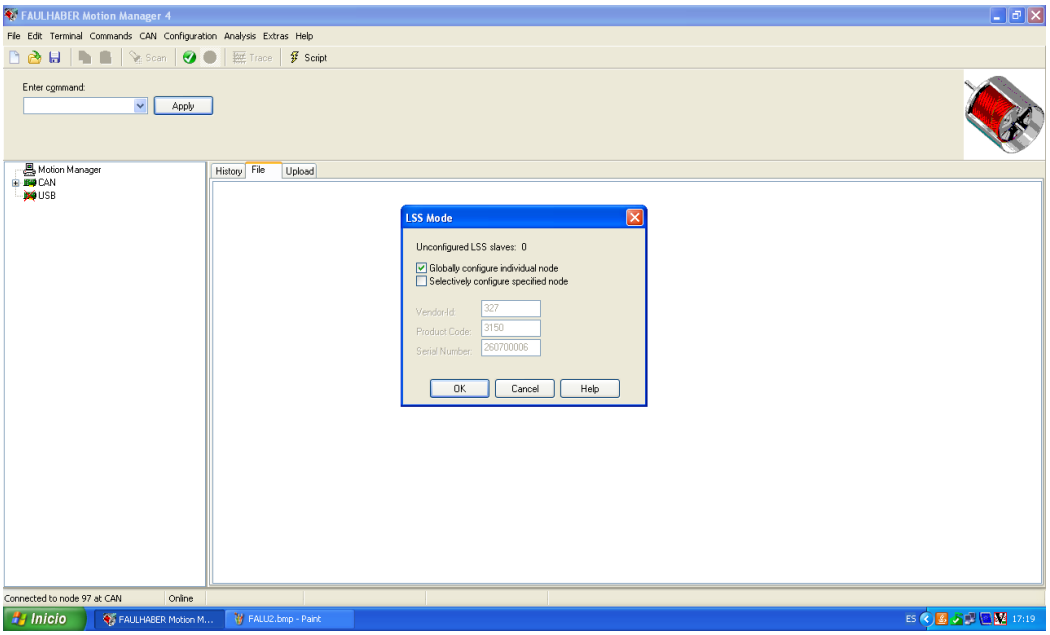


Ilustración G.1 Asignación del identificador al dispositivo MCDC3006C.

5. En la nueva ventana emergente denominada *Connection settings* que se muestra en la ilustración G.2 debe especificarse en la casilla *Baud rate* la velocidad de transmisión del bus de comunicaciones CANopen. La casilla *Node address* contiene el identificador de CANopen que se desea enviar al dispositivo MCDC3006C. El identificador debe ser un número en base decimal. Una vez elegido el valor del identificador y la velocidad de transmisión debe pulsarse el botón *Send.*

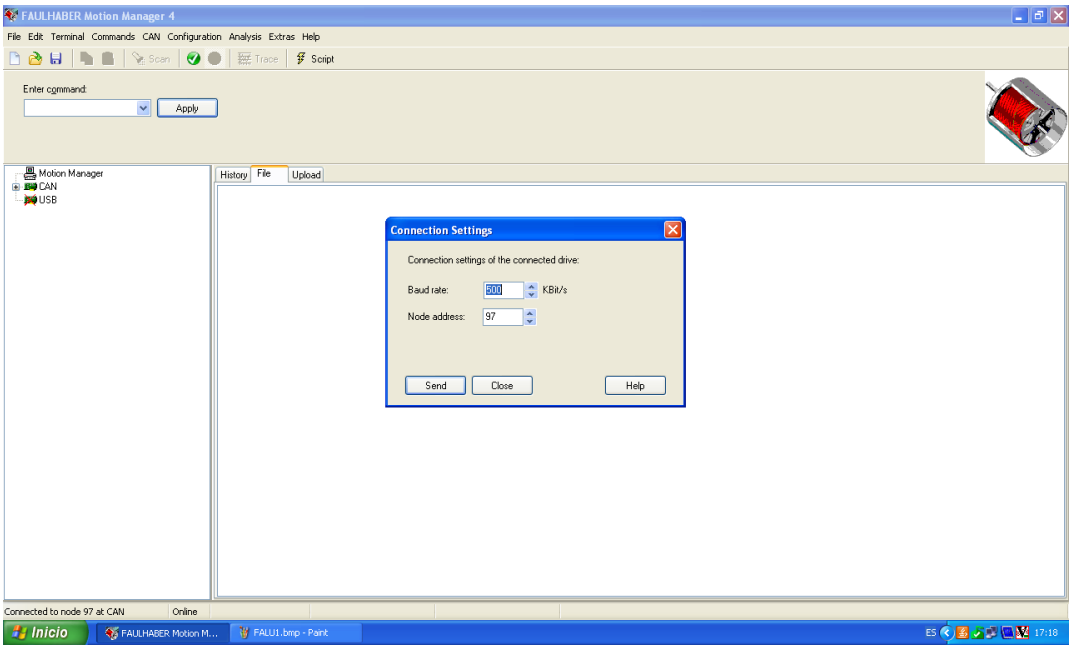


Ilustración G.2. Asignación del identificador al dispositivo MCDC3006C.